Universidade Federal do Espírito Santo

Eros Silva Spalla

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

Vitória-ES
2015

Eros Silva Spalla

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

Dissertação para obtenção do grau de mestre em Informática apresentada à Universidade Federal do Espírito Santo

Área de concentração: Ciência da Computação

Orientador: Magnos Martinello

Vitória-ES
2015

Spalla, Eros S.

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

70 páginas

Dissertação (Mestrado) - Universidade Federal do Espírito Santo.

1. SDN
2. OpenFlow
3. Resiliência

Eros Silva Spalla

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

Dissertação para obtenção do grau de Mestre em Informática apresentada à Universidade Federal do Espírito Santo. Área de concentração: Ciência da Computação.

Aprovada em 10/07/2015

Prof. Dr. Magnos Martinello
Orientador

Prof. Dr. Rodolfo da Silva Villaça
Examinador

Prof. Dr. Rafael Rodrigues Obelheiro
Examinador

Dedico este trabalho à minha família e amigos.

Agradecimentos

Agradeço ao meu orientador Magnos pelo acompanhamento, pela motivação constante, compreensão quando o tempo foi curto, e por todas as direções indicadas e ensinamentos passados ao longo do mestrado; ao professor Rodolfo por toda colaboração ao trabalho, especialmente por suas excelentes revisões; ao Christian e o Lasaro pelas valiosas discussões sobre SDN e replicação de dados.

Também não poderia esquecer de meus companheiros do NERDS, que sempre foram colaborativos e solícitos. Gostaria de agradecer especial o Alextian e o Diego, esses dois que fazem o dia ter 30 horas, por sempre terem sido os grandes parceiros de todos os momentos, desde os primeiros trabalhos das disciplinas, até a correria dos vários deadlines.

Por fim, agradeço aos meus pais, Walter e Leninha, às minhas irmãs, Jú e Mila, e à minha namorada Fran, por todos os momentos de carinho e amor que vocês me proporcionam. Apesar das distâncias, sempre mantive vocês por perto.

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 trabalho, 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* comerciais de baixo custo(RouterBoards/MikroTik) avaliando-se a latência na recuperação de falha, na migração de switches entre controladores e de processamento de *packet-in* pelos controladores. Observamos diferentes compromissos de projeto em experimentos em ambiente real sujeitos à variação de carga nos planos de dados e de controle.

Palavras-chave: SDN, OpenFlow, Resiliência, Alta-Disponibilidade.

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, switch migration and packet-in latency with different workloads. We observe a set of trade-offs in real experiments with varying workloads at both data and control plane.

Keywords: SDN, OpenFlow, Resilience, High-Availability.

Lista de Figuras

2.1 Arquitetura Fechada. 15
2.2 Arquitetura SDN [Foundation 2012]. 17
2.3 Organização do controlador Nox [Rao 2015a]. 18
2.4 Organização do controlador OpenDayLight [OpenDayLight 2015]. 19
2.5 Organização do controlador Ryu [Rao 2015b]. 19
2.6 Switch OpenFlow. 21
2.7 Arquitetura Clássica x Arquitetura OpenFlow. 23
2.8 Equipamento OpenFlow [Foundation 2012]. 24

3.1 Onix: arquitetura de controle distribuída[Koponen et al. 2010]. 28
3.2 Controlador com replicação passiva[Fonseca et al. 2013]. 29
3.3 Controlador com replicação ativa[Fonseca et al. 2013]. 30
3.4 Arquitetura Smartlight[Botelho et al. 2014]. 31

4.1 Replicação Ativa. 34
4.2 Replicação Passiva. 35
4.3 Monitoramento via agente externo. 37
4.4 Monitoramento via controladores. 38
4.5 Monitoramento via *switches*. 39

5.1 Proposta de arquitetura Resiliência para o plano de controle OpenFlow. . . 41
5.2 Open Replica para replicação e sincronização do estado. 42
5.3 Organização do controlador implementado. 44
5.4 Detecção de falha via controladores OpenFlow. 44
5.5 Processo de Recuperação de Falha. 45
5.6 Processo de Restauração da Falha. 47
5.7 Detecção de falhas via *switch* OpenFlow. 49

6.1 Recuperação e Restauração de Falha - Tráfego no Plano de Controle 54
6.2 Carga de Processamento nas RouterBoards - Tráfego no Plano de Controle 54
6.3 Latência de Detecção de Falhas pelos Controladores 55
6.4 Latência Processamento de Packet-in - Tráfego no Plano de Controle . . . 56
6.5 Recuperação e Restauração da Falha - Tráfego no Plano de Dados. 56
6.6 Carga de Processamento nas RouterBoards - Tráfego no Plano de Dados . 57
6.7 Latência de Detecção de Falhas pelos *Switches*. 58
6.8 Latência detecção e recuperação falha - *switch versus* controlador. 58
6.9 Detecção de falha - Switch x Controlador. 59

Lista de Tabelas

2.1 Tabela com o resumo das características dos principais Controladores. . . . 20
2.2 Tabela Switches 22
2.3 Tabela Evolução OpenFlow [Tourrilhes et al. 2014]. 27

6.1 Latência média em milisegundos de acordo com o número de controladores e a taxa de *packet-in/s* por controlador. 60

Lista de Abreviaturas e Siglas

BYOA Bring Your Own App

BYOD Bring Your Own Device

CAP Consistency, Availability e Partition Tolerance

CPqD Centro de Pesquisa e Desenvolvimento em Telecomunicações

CPU Computer Processor Unit

GRE Generic Routing Encapsulation

ICMP Internet Control Message Protocol

IP Internet Protocol

IPsec IP Security Protocol

IPV6 Internet Protocol Version 6

L3 Layer 3

MAC Media Access Control

MBPS Mega Bits Por Segundo

MPLS Multiprotocol Label Switching

MS Milissegundo

NAT Network Address Translation

NIB Network Information Base

OF OpenFlow

OO Orientado a Objetos

OS Operating System

OVS Open Virtual Switch

QoS Quality of Service

SCTP Stream Control Transmission Protocol

SDN Software Defined Network

TCAM Ternary Content Access Memory

TCP Transport Control Protocol

TI Tecnologia da Informação

TTL Time to Live

UDP User Datagram Protocol

VLAN Virtual Local Area Network

Sumário

1 Introdução **12**
- 1.1 Contextualização . . . 12
- 1.2 Contribuição . . . 13
- 1.3 Estrutura . . . 13

2 Fundamentação Teórica **14**
- 2.1 Limitações da Arquitetura de Rede Tradicional . . . 14
- 2.2 Arquitetura de Redes Definidas por Software . . . 16
  - 2.2.1 Plano de Controle . . . 18
  - 2.2.2 Plano de Dados . . . 20
- 2.3 OpenFlow . . . 22
  - 2.3.1 Mensagens OpenFlow . . . 24
  - 2.3.2 Evolução do OpenFlow . . . 25

3 Trabalhos Relacionados **28**
- 3.1 Onix . . . 28
- 3.2 Replicação Passiva e Ativa em SDN . . . 28
- 3.3 SmartLight . . . 30

4 Plano de Controle Resiliente **32**
- 4.1 Configuração Equal . . . 34
- 4.2 Configuração Master-Slave . . . 35
- 4.3 Configuração Multi-Master / Multi-Slave . . . 35
- 4.4 Configuração via Data Store . . . 36
- 4.5 Detecção de Falha . . . 36
  - 4.5.1 Utilizando Agentes Externos . . . 37
  - 4.5.2 Utilizando os Controladores OpenFlow . . . 37
  - 4.5.3 Utilizando os *switches* OpenFlow . . . 38
- 4.6 Considerações parciais . . . 40

5 Implementação do Protótipo **41**
- 5.1 Data Store . . . 42
- 5.2 Plano de Controle . . . 43
  - 5.2.1 Detecção de Falhas via Plano de Controle . . . 44
  - 5.2.2 Tratamento da Falha . . . 45
- 5.3 Plano de Dados . . . 47
  - 5.3.1 Detecção de Falhas via Plano de Dados . . . 48

6 Avaliação Experimental **50**
6.1 Plataforma de Avaliação 50
6.1.1 Metodologia 50
6.1.2 Carga de trabalho 51
6.1.3 Medições 51
6.2 Discussão dos Resultados 53
6.2.1 Avaliação do Ambiente Real 53
6.2.1.1 Desempenho do plano de controle 53
6.2.1.2 Desempenho do plano de dados 56
6.2.2 Avaliação Latência com Múltiplos Controladores 59
6.2.3 Comentários finais 60

7 Conclusão e Trabalhos Futuros **63**

Referências Bibliográficas **65**

Publicações do Autor **70**

Capítulo 1

Introdução

1.1 Contextualização

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 aos dispositivos, 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.

1.2 Contribuição

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 entre 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 observadas 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 a maior parte dos trabalhos relacionados apenas consideraram ambientes emulados.

Nossa terceira contribuição diz respeito ao modo de detecção de falhas dos controladores. E neste caso as especificações do OpenFlow delegam essa função aos projetistas de redes, apesar da versão 1.5 apresentar uma nova *feature* que possibilita aos controladores receberem eventos sobre o estado da conexão de outros controladores da rede. Desse modo, propomos uma nova técnica de detecção de falha, em que os controladores são notificados a partir de um mecanismo implementado nos *switches*.

1.3 Estrutura

O restante deste trabalho está estruturado da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica, o objetivo deste capítulo é discutir aspectos teóricos importantes para compreensão do trabalho, deste modo, discutimos os principais conceitos sobre redes SDN e o protocolo OpenFlow, evidenciando os elementos básicos para um ambiente SDN. Há uma miríade de questões de pesquisa em recentes trabalhos relacionados à distribuição de controle em redes SDN objetivando soluções de alta disponibilidade, todavia, no Capítulo 3 citamos apenas os trabalhos mais próximos ao foco da proposta deste trabalho, que é a solução do problema de conectividade com múltiplos controladores e o uso de *data stores* para consistência do estado. O Capítulo 4 discute diferentes abordagens para implementação de resiliência em controladores OpenFlow. O Capítulo 5 apresenta uma proposta para controladores resilientes. O Capítulo 6 lista os procedimentos metodológicos, expõe os resultados obtidos e aponta o comportamento do experimento em ambiente real. Por fim, no Capítulo 7 são tecidas as considerações finais e a direção dos trabalhos futuros.

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

Capítulo 2

Fundamentação Teórica

2.1 Limitações da Arquitetura de Rede Tradicional

Observando a infraestrutura de comunicação de dados existente, podemos afirmar que as redes de computadores se tornaram um ponto crítico em praticamente todas as áreas de negócio que conhecemos. Existem milhares de aplicações que funcionam sobre um estrutura de *switches*, roteadores, *bridges*, *firewalls* e etc., que necessitam de características como: confiabilidade, segurança, baixa latência, flexibilidade, e escalabilidade, para citar algumas. Entretanto, apesar do aumento das capacidades dos canais de comunicação e maior poder de processamento dos equipamentos de rede permitirem o surgimento de uma grande quantidade de aplicações ([Ghemawat et al. 2003], [DeCandia et al. 2007],[Dean and Ghemawat 2008 [Chang et al. 2008]), a rede em sua infraestrutura, do ponto de vista arquitetural, não apresentou tantas mudanças.

Desde o momento em que as redes passaram a empregar equipamentos como *switches* e roteadores para encaminhar pacotes, a estrutura pouco se alterou. Fabricantes criam equipamentos como uma caixa-pretas, de modo que o *firmware* esteja encapsulado e fechado no *hardware*, fundindo-os como se fossem uma entidade única. E então, empresas constroem suas redes como um aglomerado de caixas-preta interligadas.

Este modelo estabelecido é frequentemente referenciado, do ponto de vista estrutural, como uma benção e uma maldição. A forma como a evolução da rede se deu, criou como base um emaranhado de protocolos e camadas de equipamentos, que apesar de permitir a comunicação da forma como conhecemos hoje, ergueu-se uma enorme barreira para inovações e experimentações necessárias à área de redes de computadores. Qualquer inovação nessa infraestrutura representa um processo de alta complexidade. Segundo Nick Mckeown[McKeown 2009], pouca tecnologia é transferida da academia para o mercado, e em geral, a partir de sua concepção, uma inovação leva até 10 anos para entrar em produção.

Tradicionalmente, esse modelo impõe diversas barreiras para os administradores de

redes. Por exemplo, se o administrador de redes desejar realizar alguma alteração na lógica de tratamento/encaminhamento e/ou roteamento dos pacotes, o equipamento não permite, pois apenas uma interface de alto nível é disponibilizada para configuração do *hardware*. E assim, normalmente, a adição de novas *features* significa novos produtos ou novas licenças de *software*, o que cria uma profunda dependência dos fabricantes de *hardware*. A Figura 2.1 apresenta um exemplo de equipamentos com arquitetura fechada.

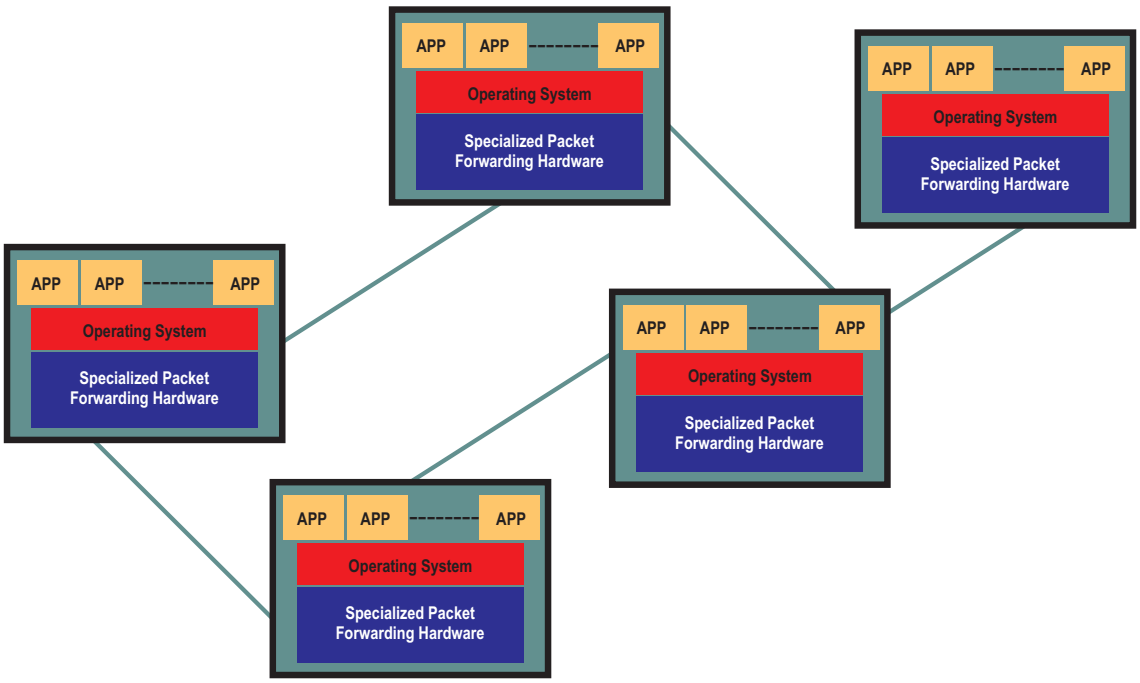


Figura 2.1: Arquitetura Fechada.

A seguir, destacamos alguns pontos que explicitam a necessidade de um novo paradigma de redes em face às demandas atuais:

1. Novos padrões de tráfego: atualmente, com os grandes *Data Centers* de empresas que provêm serviços como computação na nuvem, redes centradas em conteúdo, e outros, os padrões de tráfego mudaram sensivelmente. Ao contrário de uma comunicação cliente-servidor tradicional, em que um cliente conecta-se a um servidor em uma comunicação "vertical", as novas aplicações utilizam diferentes bancos de dados e servidores, o que significa que para a requisição, esse fluxo de dados irá envolver diversos pontos da rede antes de retornar ao usuário. Também, a forma como o usuário se conecta e interage como a rede, seja via um *smartphone*, *tablet* ou computador altera esses padrões de tráfego, já que cada um desses equipamentos tem sua forma de consumir e gerar dados;

2. consumerização de TI: usuários cada vez mais fazem acesso às redes corporativas a partir de diferentes dispositivos como *smartphones* e *tablets*. Desse modo, existe uma necessidade crescente de configurações específicas de acordo com o perfil dos

usuários, a fim de permitir o acesso à rede, ao mesmo que tempo em que os níveis de segurança sejam mantidos. Essa mudança, onde *bring your own device* (BYOD) e *bring your own app* (BYOA) foram os principais agentes transformadores, traz um grande desafio para a gerência de TI, já que são comportamentos que estão em plena evolução e condizem com a perspectiva atual de TI, em que a tecnologia está cada vez mais presente, intuitiva e *user-driven*.

3. Computação na nuvem e virtualização: com o advento da computação na nuvem, empresas vislumbraram diversas possibilidades para seus negócios, o que resultou em um crescimento expressivo das soluções em nuvem. Todavia, manter uma infraestrutura do tipo *cloud* envolve requisitos de segurança, disponibilidade, escalabilidade, auditoria, mudanças das regras de negócio, e grande volume de dados, o que significa uma estrutura complexa. Entretanto, ainda assim, é desejável que todo o gerenciamento da infraestrura seja simples.

2.2 Arquitetura de Redes Definidas por Software

As Redes Definidas por Software surgiram como um promissor conceito para o projeto de novas arquiteturas para a Internet [McKeown 2009]. O ponto central das SDN encontra-se na separação dos planos de dados e de controle em uma interface uniforme, independente de fornecedor, para o mecanismo de encaminhamento (ex.: OpenFlow [McKeown et al. 2008]).

Lantz [Lantz et al. 2010] explica que em uma rede definida por *software* o plano de controle (ou "sistema operacional de rede") é separado do plano de dados. Normalmente, o sistema operacional de rede observa e controla o estado de toda a rede a partir de um ponto central, oferecendo recursos como: protocolos de roteamento, controle de acesso, virtualização de rede, gestão de energia e também prototipação de novos protocolos. A principal consequência das SDN é que as funcionalidades da rede podem ser facilmente alteradas ou mesmo definidas após a rede ter sido implantada. Novas funcionalidades podem ser adicionadas, sem a necessidade de se modificar o *hardware*, permitindo que o comportamento da rede evolua na mesma velocidade que o *software*.

Entretanto, o aspecto mais importante de uma rede SDN é o alto grau de flexibilidade oferecido pela programabilidade da rede, uma vez que os projetistas de redes têm a possibilidade de configurar e escrever suas aplicações por meio de uma camada de abstração de alto nível, em vez de aguardar um *update* de *firmware* ou um novo produto do fabricante. E mais, isso permite que a cada alteração na lógica de encaminhamento da rede ou adição de uma nova *feature*, o código da aplicação seja simplesmente atualizado. Como a camada de aplicação independe do *hardware*, a evolução da aplicação pode ser dinâmica e constante. Assim, todo o poder que uma linguagem de programação pode oferecer, passa

a estar disponível para ser utilizada em uma aplicação SDN.

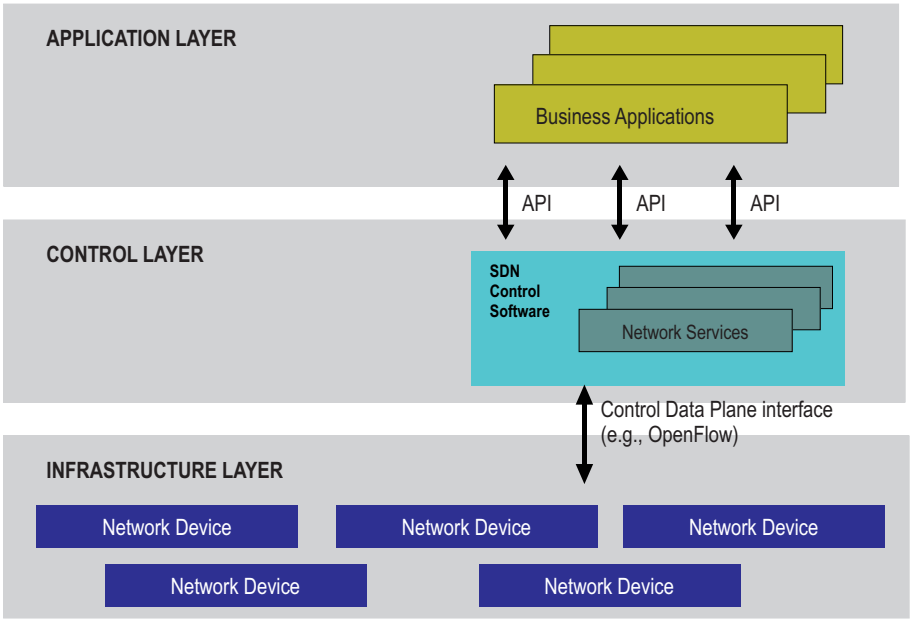


Figura 2.2: Arquitetura SDN [Foundation 2012].

A Figura 2.2 apresenta a visão lógica da arquitetura SDN. A inteligência da rede é centralizada (logicamente) em controladores SDN, os quais mantêm a visão global dos estados da rede. Como resultado dessa abstração, para a camada de aplicação, a rede é apresentada como um único *switch* lógico. Como a visão da rede é logicamente centralizada, a implementação e operação da rede torna-se mais simples, assim como o plano de dados. Como equipamentos de encaminhamento passam a ter uma função muito bem definida, que é encaminhar pacotes, isso permite que o *hardware* também possa ser mais simples, o que pode resultar em um equipamento com um custo mais baixo.

Abaixo, listamos algumas características inerentes ao uso de SDN's [Foundation 2012]:

- Gerenciamento e controle centralizado dos equipamentos de múltiplos fabricantes;
- Maior capacidade de inovação, já que que novas ferramentas e serviços podem surgir por meio de aplicações desenvolvidas, ou seja, por desenvolvimento de *software*, sem a necessidade fazer alterações no *hardware*;
- Controle granular da rede, com possibilidade de aplicar diferentes políticas de segurança a diferentes perfis de usuários e equipamentos;
- Melhoria da automação do gerenciamento, usando diferentes APIs para abstrair os detalhes da rede para outras aplicações, como sistemas de orquestração, sistemas de provisionamento e aplicações de rede.

Nas próximas duas subsecções apresentaremos pontos relevantes da arquitetura SDN, em especial os planos de controle e dados, além de uma descrição do protocolo OpenFlow. Essas informações serão úteis para compreensão da arquitetura proposta.

2.2.1 Plano de Controle

O plano de controle de uma rede SDN é a abstração que encapsula os conceitos relacionados ao controle de uma rede de computadores convencional. Com essa nova arquitetura, em que o controle ou a inteligência da rede é desacoplada da camada física dos equipamentos, isto é, do plano de dados, surge o sistema operacional de redes. Basicamente, os sistemas operacionais fornecem um mecanismos para simplificar a interação do plano de controle com o plano de dados, e uma interface para que aplicações possam ser desenvolvidas. Por exemplo, um projetista de redes ao escolher uma plataforma *network OS (Operating System)*, passa a ter uma visão de alto do nível do plano de controle, desta forma, programar uma rede SDN torna-se uma tarefa muito parecida como desenvolver uma aplicação em uma linguagem de programação qualquer.

Do mesmo modo como em outras arquiteturas, que existem diversos sistemas operacionais, em SDN's não é diferente. Controladores são desenvolvidos cada um com características e objetivos bastante variados, cobrindo desde o controle de ambientes para experimentações acadêmicas, até redes de grande porte. A seguir, listamos alguns controladores e suas principais características:

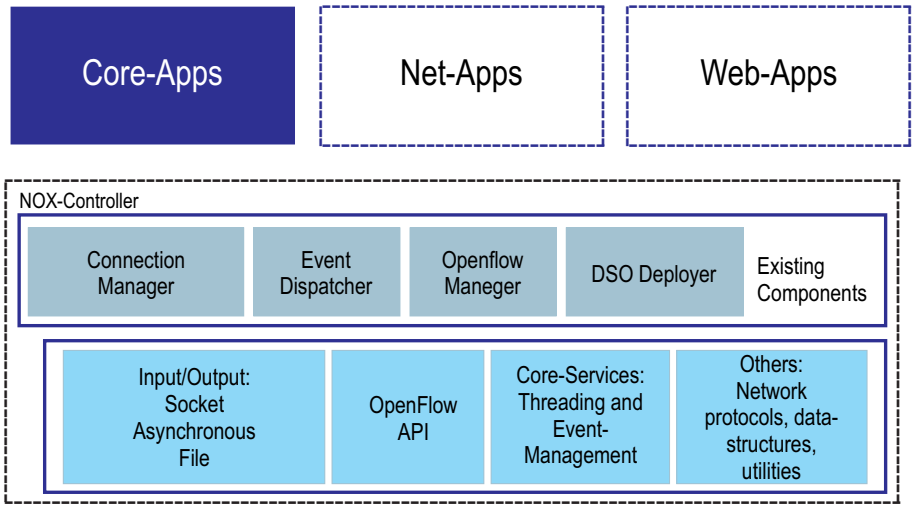


Figura 2.3: Organização do controlador Nox [Rao 2015a].

Nox [Gude et al. 2008] é um controlador OpenFlow desenvolvido inicialmente pela Nicira® e então, a partir de 2008, disponibilizado para a comunidade. A interface de programação disponível para criar aplicações, assim como a linguagem utilizada no desenvolvimento do controlador em questão, foi o C++. Uma das principais características deste controlador é o alto desempenho. Atualmente, possui suporte à versão 1.0 do OpenFlow, entretanto existe uma versão modificada pelo CPqD (Centro de Pesquisa e Desenvolvimento em Telecomunicações) que suporta parcialmente a versão 1.3 [Fernandes 2013]. A Figura 2.3 apresenta a arquitetura do controlador Nox.

A partir do controlador Nox surgiu um novo projeto, cujo objetivo prover uma interface mais simples para controladores SDN. Desenvolvido em Python, o controlador Pox **??** é

uma versão do controlador Nox tradicional. Este controlador normalmente é utilizado como uma alternativa ao Nox em experimentos e prototipação, já que possui uma interface mais amigável.

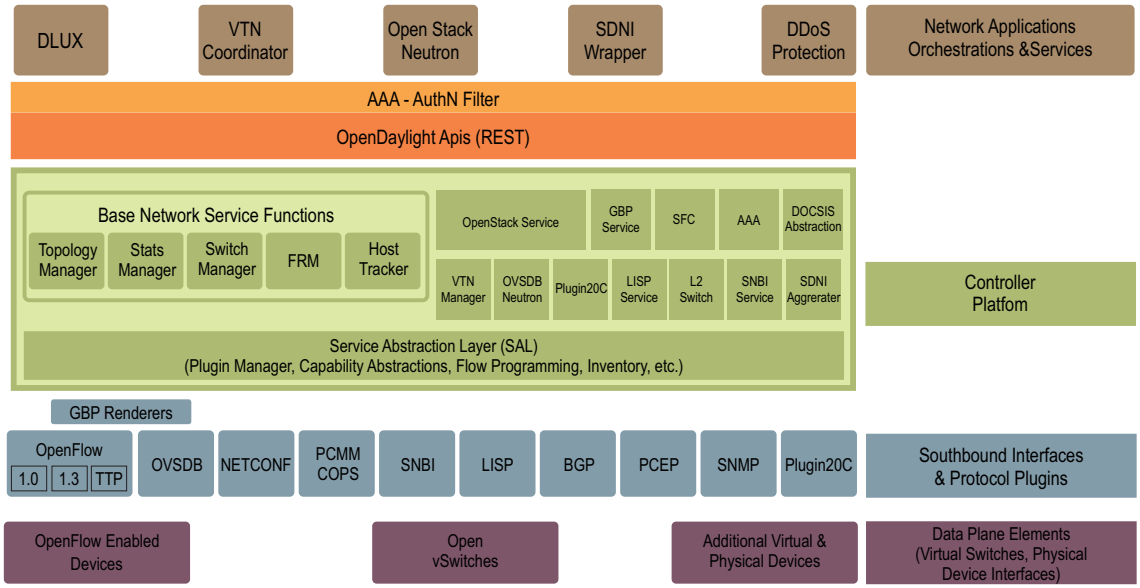


Figura 2.4: Organização do controlador OpenDayLight [OpenDayLight 2015].

OpenDayLight [OpenDaylight 2013] é um projeto *open source* cujo desenvolvimento tem participação de grandes empresas como Cisco®, Citrix®, Microsoft®, IBM®, dentre outras. O principal objetivo da comunidade é acelerar o processo de popularização do uso de SDN's, e como parte do projeto, disponibilizam a plataforma para o desenvolvimento de aplicações em SDN's. O OpenFlow é um dos padrões suportados pelo OpenDaylight, e atualmente tem suporte às versões 1.0 e 1.3 do OpenFlow. A Figura 2.4 ilustra a organização do controlador OpenDayLight.

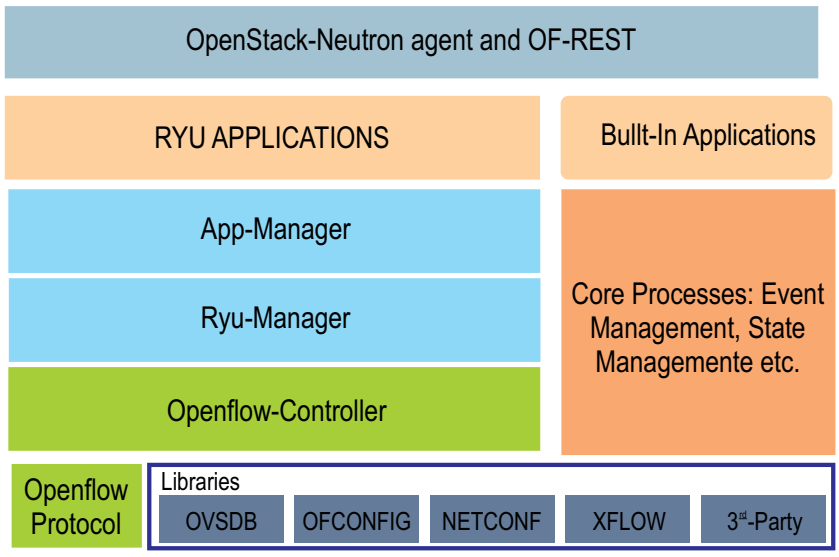


Figura 2.5: Organização do controlador Ryu [Rao 2015b].

Ryu [RYU 2014] é um controlador *open source* desenvolvido por um grupo japonês da NTT Lab's. O projeto é totalmente implementado em Python, e possui boa integração com outras ferramentas de rede, como por exemplo o OpenStack [Corradi et al. 2012]. Um dos principais pontos do projeto é o suporte a vários protocolos de *southbound*, como OpenFlow, NetConf e OF-Config. O desenvolvimento constante por parte da comunidade permite que o controlador esteja atualizado com a versões mais recentes do Openflow. Atualmente, possui suporte total até a versão 1.4, e algumas implementações parciais da versão 1.5. A arquitetura do controlador japonês Ryu é apresentada na Figura 2.5.

A Tabela 2.1 sintetiza o resumo dos principais controladores, juntamente com informações adicionais, como a versão do OpenFlow suportado e a linguagem da API.

Tabela 2.1: Tabela com o resumo das características dos principais Controladores.

Controlador	**Suporte OpenFlow**	**Licença**	**Linguagem API**
Nox	1.0	GPLv3	C++
Pox	1.0	GPLv3	Python
OpenDayLight	1.0 e 1.3	EPL v1.0	Java
Ryu	1.0, 1.1, 1.2, 1.3 e 1.4	Apache 2.0	Python

2.2.2 Plano de Dados

O plano de dados ou plano de encaminhamento em uma rede definida por *software* tem como função central o encaminhamento de pacotes, isto é, escoar o tráfego da rede. Logo, o plano de dados passa a ser representado pelos equipamentos de rede, sejam eles físicos ou virtuais, que possuam a capacidade de encaminhar pacotes.

De modo geral, os equipamentos de encaminhamento tradicionais possuem uma ou mais tabelas de fluxos, normalmente em porções de memória TCAM (*Ternary Content Access Memory*), onde são implementados serviços de *firewall*, NAT (*Network Address Translation*), *traffic shaping*, roteamento, entre outros. Vale lembrar que, no paradigma de rede tradicional, como a arquitetura é fechada, cada fabricante faz essas implementações de modo distinto. Em equipamentos OpenFlow, por ser *open-source*, o conjunto de instruções do protocolo pode ser implementado em diferentes equipamentos, já que a tabela de fluxos é definida pelo protocolo.

Um *switch* OF (OpenFlow), também chamado *datapath*, pode ser representado em três partes:

1. Tabela de Fluxos: nessa tabela, cada fluxo é associado a uma ação, e assim, os pacotes que chegam ao equipamento são encaminhados de acordo com as regras instaladas na tabela de fluxos;

2. Canal Seguro: esse canal conecta o equipamento ao controlador remoto, utilizando uma conexão criptografada, e permite que mensagens e pacotes sejam enviadas entre o controlador e o *switch*;

3. O protocolo OpenFlow: o protocolo define um padrão aberto de comunicação entre o controlador e o *switch*, que permite a programação da tabela de fluxos do equipamento por meio de uma interface de alto nível. Assim, o projetista de redes não precisa se preocupar com as características do equipamento.

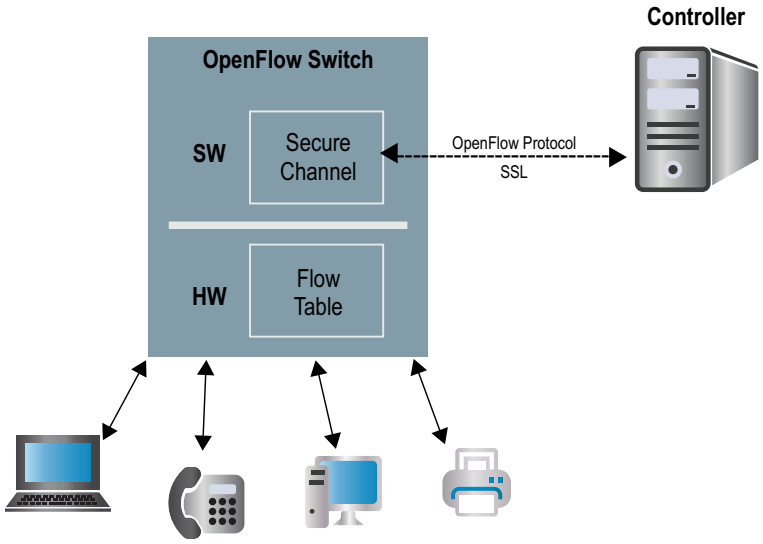


Figura 2.6: Switch OpenFlow.

Entre os diversos *switches* OpenFlow, que podem ser físicos ou virtuais, podemos citar:

1. Open vSwitch [Pfaff et al. 2009] é um *switch* virtual *open-source* que segue a arquitetura OpenFlow. Uma característica importante deste encaminhador é que, apesar de ser implementado em *software*, seu plano de dados é executado em *kernel mode*, enquanto o plano de controle é acessado a partir do espaço de usuário. O OvS (Open Virtual Switch) possui suporte completo a versão 1.3 do OpenFlow.

2. Pica8 é um *switch* OpenFlow de alto desempenho baseado em Linux. Fabricado pela empresa Pica8 Inc., este equipamento executa um sistema operacional próprio, chamado PicOS, que inclui capacidade de encaminhamento *layer-2* e *layer-3*, e suporte ao protocolo OpenFlow através de uma implementação do Open Vswitch embarcada no sistema. O sistema PicOS é um sistema operacional para equipamentos de rede, baseado no projeto XORP [Handley et al. 2005], que é executado em um kernel Linux, com a adição de serviços de rede. Atualmente possui suporte completo a versão 1.4 do OpenFlow.

3. Plataforma aberta programável [Liberato et al. 2014] é um projeto do NERDS (Núcleo de Estudos em Redes Definidas por Software), grupo de pesquisa ligado à UFES

(Universidade Federal do Espírito Santo), que desenvolve pesquisas em SDN e possui um projeto que estuda plataformas de *hardware* que possam ser utilizadas como encaminhadores OpenFlow. Neste projeto, equipamentos do tipo RouterBoard da empresa Mikrotik basicamente têm seu sistema operacional substituído pelo sistema *open source* OpenWRT, junto com uma versão do OvS compilada especialmente para essa arquitetura. No momento, o projeto suporta a versão 1.3 do OpenFlow.

4. CPqD Softswitch13 [CPqD 2014], assim como o OvS, é um *switch open source* implementado em *software*. Todavia, neste projeto, tanto o acesso ao plano de controle quanto o plano de dados, são implementados no espaço do usuário. O Softswitch13 é baseado nos projeto Ericsson's Traffic Lab OpenFlow 1.1 *switch*[1] e no Stanford OpenFlow 1.0 reference switch[2], com alterações para suportar a versão 1.3 do OpenFlow.

A Tabela 2.2 apresenta uma visão sistemática dos encaminhadores citados.

Tabela 2.2: Tabela Switches

Switch	**Suporte OpenFlow**	**Licença**	**Modo de Encaminhamento**
Open Vswitch	1.0, 1.1, 1.2 e 1.3	Apache 2.0	Kernel
Pic8	1.0, 1.1, 1.2, 1.3, 1.4	Copyright	Kernel
Plataforma Aberta	1.0, 1.1, 1.2 e 1.3	Apache 2.0	Kernel
CPqD Softswitch13	1.0 e 1.3	BSD	Espaço de usuário

2.3 OpenFlow

O protocolo OpenFlow foi o primeiro padrão de interface de comunicação entre o plano de controle e o plano de dados de uma arquitetura SDN [Foundation 2012]. O OpenFlow permite o acesso e manipulação do plano de dados nos equipamentos de rede como roteadores e *switches*, tanto fisica quanto virtualmente (baseado em *hypervisor*). Essa interface padronizada e aberta permite que o plano de controle possa se comunicar com o plano de dados independentemente do fabricante. Essa característica é exatamente oposta ao que existia até então, equipamentos de redes com o sistema fechado em uma arquitetura monolítica, e extremamente dependente dos fabricantes.

Sherwood [Sherwood et al. 2009] apresentou uma comparação entre a arquitetura tradicional e a arquitetura que utiliza o protocolo OpenFlow. Na arquitetura clássica, tanto

[1] https://github.com/TrafficLab/of11softswitch
[2] http://yuba.stanford.edu/git/gitweb.cgi?p=openflow.git;a=summary

a lógica de controle quanto a lógica de encaminhamento estão localizadas internamente ao *switch*, comunicando-se através de um barramento proprietário do fabricante. Na arquitetura OpenFlow o *switch* executa apenas a lógica de encaminhamento. A lógica de controle é executada no controlador OpenFlow e a comunicação entre elas se dá através do protocolo OpenFlow. A Figura 2.7 ilustra as duas arquiteturas.

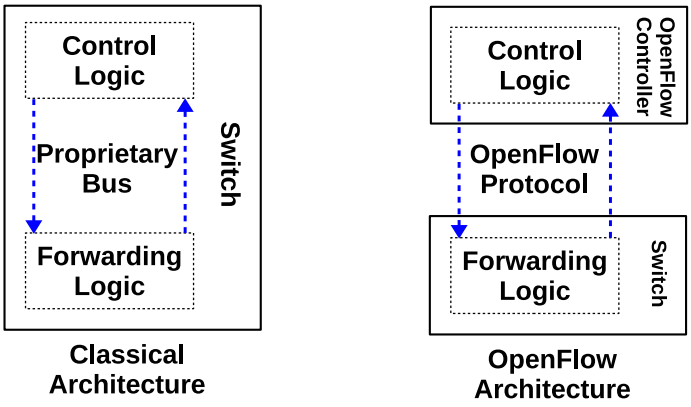
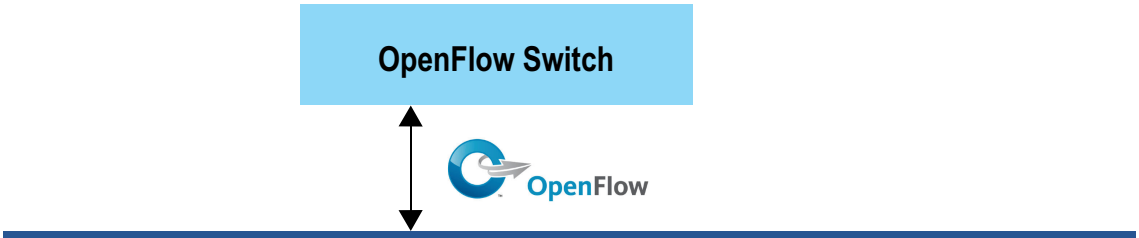


Figura 2.7: Arquitetura Clássica x Arquitetura OpenFlow.

O OpenFlow utiliza o conceito de fluxos para identificar o tráfego da rede, que são representados como um conjunto de pacotes que possuem campos dos cabeçalho de dados com o mesmo valor. Assim, um equipamento, ao receber o trafégo da rede, ou seja, fluxos, consulta a sua tabela de fluxos, definida pela aplicação da rede, de modo a decidir como encaminhar estes pacotes. Essas regras podem ser definidas dinâmica ou estaticamente. Caso não haja regras que coincidam com o fluxo recém chegado, o primeiro pacote do fluxo (*packet-in*) é encaminhado ao controlador, que então realiza o processamento necessário para definir uma nova regra de encaminhamento para tal fluxo. A flexibilidade dessa abordagem permite aos projetistas de redes definir como os pacotes serão encaminhados pelos equipamentos baseando-se em parâmetros, padrões de tráfego e aplicações, por exemplo.

De acordo com o apresentado na Figura 2.8, o protocolo define um conjunto de operações que podem ser utilizadas por uma aplicação do plano de controle para programar o plano de dados, por exemplo como *learning switch*. O conjunto de operações disponibilizadas pelo OpenFlow está para o protocolo assim como o conjunto de instruções está para uma arquitetura de processador.

Uma rede SDN que utiliza o protocolo OF é basicamente composta por um ou mais controladores, aplicações, e equipamentos que operam o plano de dados, como roteadores e *switches*.



OpenFlow Switch-enabled Network Device

Flow Table Comparable to an instruction set

MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:.	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192*	*	*	local	120
*	*	*	*	*	*	controller	11

Figura 2.8: Equipamento OpenFlow [Foundation 2012].

2.3.1 Mensagens OpenFlow

Controladores OpenFlow se comunicam com o plano de dados por meio de mensagens OF. Nesta seção, apresentamos algumas das principais mensagens OpenFlow, que podem ser classificadas como síncronas ou assíncronas. Para uma descrição mais detalhada do protocolo, a especificação oficial pode ser consultada [McKeown et al. 2008].

As mensagens síncronas são caracterizadas por serem mensagens iniciadas pelo controlador e normalmente exigirem uma resposta ou confirmação. Dentre estas, podemos citar:

- *Features-Request* - Uma vez estabelecida a conexão entre o controlador e o *switch*, o controlador envia uma mensagem de requisição ao *switch*. O *switch* responde a solicitação informando os dados referentes as suas capacidades e configurações;
- *Flow-Mod* - Mensagens deste tipo são enviadas pelo controlador para gerenciar os estados do *switch*. O propósito dessa instrução é adicionar, modificar e excluir fluxos nas tabelas dos equipamentos;
- *Stats-Request* - Essas requisições são utilizadas pelo controlador para coletar estatísticas sobre as tabelas, portas e fluxos dos *switches*;
- *Barrier-Request* - Mensagens de barreira são utilizadas pelo controlador para garantir que o processamento de pacotes pelo *switch* seja realizado de acordo com uma sequência ou para confirmar que determinada operação foi executada. Por exemplo,

caso existam 5 operações pendentes em um *switch* e uma *barrier-request* é enviada, isso significa que todas as operações pendentes devem ser processadas antes do que qualquer outro fluxo que tenha chegado após a mensagem de barreira.

Os equipamentos do plano de dados, em geral, executam ações indicadas pelo controlador, entretanto, em algumas situações, é preciso que mensagens sejam enviadas a partir do plano de dados. Essas mensagens são chamadas de assíncronas, e têm como objetivo informar os controladores sobre eventos do plano de dados, como chegada de pacotes, mudança de estado no *switch* ou a ocorrência de um erro. Abaixo são descritas algumas das principais mensagens assíncronas.

- *Packet-In* - Sempre que um novo fluxo chega ao *switch* e não há uma entrada na tabela de fluxos que faça *matching* com esse conjunto de pacotes, uma mensagem do tipo *packet-in* é enviada ao controlador. Como a lógica de encaminhamento está localizada no plano de controle, cabe ao controlador processar tal pacotes e definir qual encaminhamento este fluxo terá;

- *Port-Status* - Essa mensagem informa ao controlador mudanças que ocorram na portas do *switch*. Por exemplo, caso um cabo seja desconectado ou uma *interface* esteja desabilitada, uma mensagem *port-status* será enviada ao plano de controle;

- *Flow-Removed* - Quando uma regra é instalada na tabela de fluxos de um *switch*, deve-se indicar quando essa regra irá expirar, seja por inatividade ou por tempo. Assim que uma regra é removida da tabela de fluxo, todos os controladores recebem uma mensagem do tipo *flow-removed*.

2.3.2 Evolução do OpenFlow

As primeiras versões do protocolo OpenFlow datam de meados de 2008, entretanto, somente ao fim de 2009 uma versão estável foi lançada. Na versão 1.0 [Specification 2009] o controlador dispõe de uma única tabela de fluxo em cada switch, e o *match* pode ser feito sobre os campos dos protocolos Ethernet, VLAN, IP, ICMP, TCP e UDP. Uma outra importante *feature* é o *slicing*, que é um mecanismo de QoS que permite isolamento de tráfego em redes OpenFlow. Essa versão suporta ainda máscaras de bits nos campos do protocolo IP.

Na versão 1.1 do protocolo OpenFlow [Specification 2011a] o controlador passa a suportar operações em múltiplas tabelas de fluxo, além de um grande conjunto de novas ações (*copy/decrement* TTL, *push/pop* tag, QoS, *Groups actions*) e campos de cabeçalho (VLAN *priority*, MPLS *tag*), o que permite aumentar a flexibilidade no uso do protocolo.

No fim de 2011 foi lançada a versão 1.2 do OpenFlow [Specification 2011b], que por meio do conceito de papéis passa a suportar a conexão dos *switches* a múltiplos controladores. Nesta versão, outras *features* que merecem destaque são: suporte ao protocolo

IPV6, *multi layer switching*, túneis GRE/L3, possibilidade de uso de SCTP e UDP+IPsec para a conexão de controle, entre outros.

A versão 1.3 [Specification 2012], lançada em abril de 2012, traz suporte aos cabeçalhos de extensão do IPV6, *meters* de tráfego por fluxo e QoS por aplicação. Esta versão está sendo amplamente aceita pela comunidade, no sentido de que mostrou-se que o desenvolvimento do protocolo está alinhado às demandas das redes atuais.

Em 2013 foi lançada a versão 1.4 [Specification 2013], cujas principais inovações desta versão são: suporte a execução de um conjunto de ações (*bundles*) por meio de uma única operação; *vacancy events*, que sinalizam ao plano de controle caso a tabela de fluxos esteja próxima de seu limite; *flow monitors*, que permitem os controladores monitorarem alterações nos *switches*; e notificação dos controladores em caso de troca de papel para os *switches*. Também, a porta padrão para o protocolo foi alterada, passando de 6633 para 6653. Essa versão trouxe uma quantidade considerável de novas características em relação à versão anterior.

Por último, a versão 1.5 [ONF 2014], lançada em 2014, traz mais de 20 novas *features*, entre as principais, suporte a *egress tables*, monitoramento das conexões com os controladores, e *pipeline* baseado no tipo do pacote. Em versões anteriores à 1.5 todos os pacotes processados deviam ser do tipo *Ethernet*, com a inclusão de *packet type aware pipeline*, passa a ser possível o processamento de outros tipos de pacotes, como IP ou PPP.

A Tabela 2.3 sintetiza as mudanças ocorridas no OpenFlow ao longo das versões do protocolo.

Tabela 2.3: Tabela Evolução OpenFlow [Tourrilhes et al. 2014].

Agrupamento dos Recursos	**Descrição dos Recursos**	**Versão OpenFlow Suportada**
Separação de Planos	*Flow cookies* - Política para identificar entradas de fluxos	1.0
	Vacancy Events - Notificação sobre o esgotamento de recursos	1.4
Visão Centralizada	Papéis OpenFlow (Multi-controlador)	1.2
	Event-filtering por conexão (Multi-Controlador)	1.2
	Bundles - Aplicar um conjunto de operações em uma única requisição	1.4
	Connection status - Monitoramento das conexões dos controladores	1.5
	Monitoramento de fluxos (Multi-controlador)	1.4
Northbound API's	*Slicing* - isolamento de tráfego	1.0
	Meters por fluxo	1.3
	Tunnel id - identificação do encapsulamento utilizado	1.3
Programação	*Groups* Operações sobre grupos *flow actions*	1.0
	Extensão de campos dos pacotes e reescrita de cabeçalho	1.2
	Table Features - Maior flexibilidade na operação de *flow miss*	1.3
Tabelas de Fluxos	Múltiplas tabelas de fluxo	1.1
	Egress tables - processamento de pacotes baseado no contexto de porta de saída	1.5
	Pipeline baseado no tipo de pacote	1.5

Capítulo 3

Trabalhos Relacionados

3.1 Onix

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. A Figura 3.1 ilustra a organização implementação apresentada por Koponen.

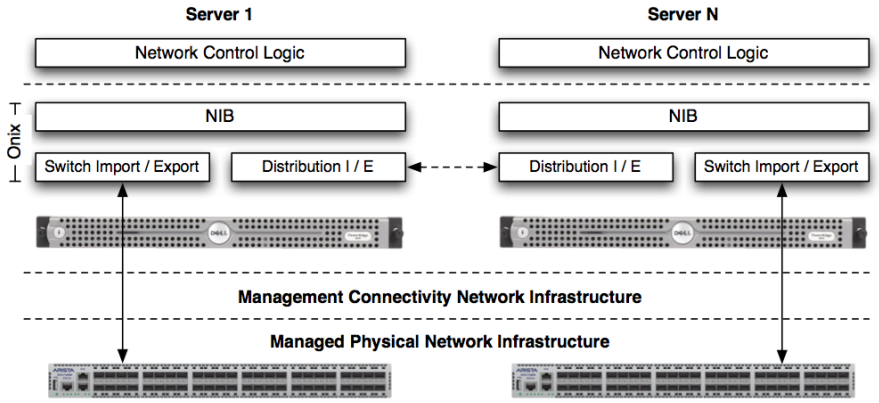


Figura 3.1: Onix: arquitetura de controle distribuída[Koponen et al. 2010].

3.2 Replicação Passiva e Ativa em SDN

Em [Fonseca et al. 2013] os autores apresentam um estudo em SDN em que as estratégias de replicação 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 um controlador

primário, que faz o processamento de todos pacotes da rede, enquanto podem existir inúmeros controladores secundários, que não processam nenhum pacote dos controladores. Nesta técnica, todo o estado do controlador primário é replicado nos controladores secundários, ou seja, o controlador primário, à medida que recebe e processa o pacotes dos *switches*, envia mensagens aos controladores secundários atualizando seus estados.

Para o tratamento de falhas, caso o controlador primário esteja *down*, os *switches* acessam uma lista de controladores, previamente configurada, e se conectam a um novo controlador. Este controlador passa a ser o novo controlador primário, com a garantia de ter seu estado consistente, e a rede continua em operação após o período de falha. A Figura 3.2 ilustra a implementação da estratégia de replicação passiva apresentada por Fonseca.

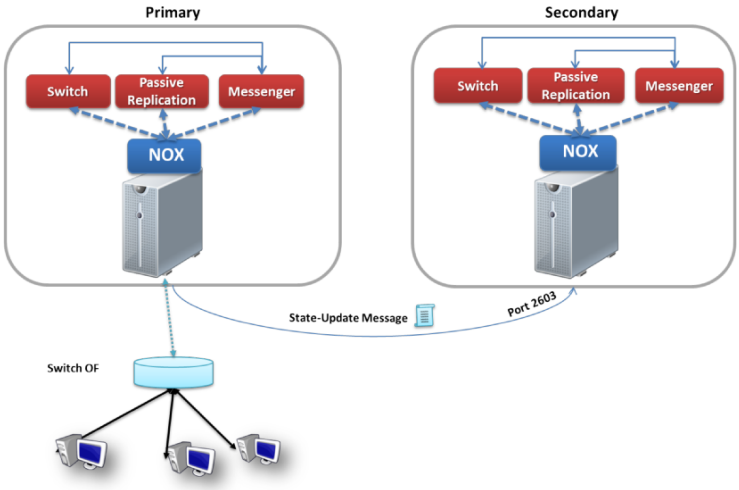


Figura 3.2: Controlador com replicação passiva[Fonseca et al. 2013].

Na implementação em que utiliza-se replicação ativa, os *switches* se conectam a todos os controladores da rede, e estes processam os pacotes de forma simultânea. Neste método, ao enviar um pacote para o plano de controle, todos os controladores recebem e processam os pacotes dos *switches*, e então, apenas um controlador responde tal requisição, e para tanto, trocam mensagem de controle para definir qual controlador processou o pacote em menor tempo. Essa técnica permite que os estados da rede estejam replicados em cada controlador, já que todos controladores processam o mesmo pacote. A Figura 3.3 ilustra a implementação da estratégia de replicação ativa apresentada por Fonseca.

O procedimento para tratamento de falhas é característica intríseca à arquitetura proposta. Como todos os controladores da rede processam os mesmos pacotes, o que os leva a ter os mesmos estados, caso um controlador não esteja mais disponível, esta falha não será perceptível, desde que haja pelo menos mais um controlador na rede.

Segundo os autores, a implementação da estratégia de replicação passiva mostra-se mais indicada para ambientes que utilizem uma abordagem menos intrusiva e que tenham menor sensibilidade à indisponibilidade da rede. Já a técnica de replicação ativa dos

controladores tem indicação para ambiente com baixa tolerância a falhas, já que neste caso, por conta dos múltiplos controladores que processam os pacotes da rede, não há período de *downtime*.

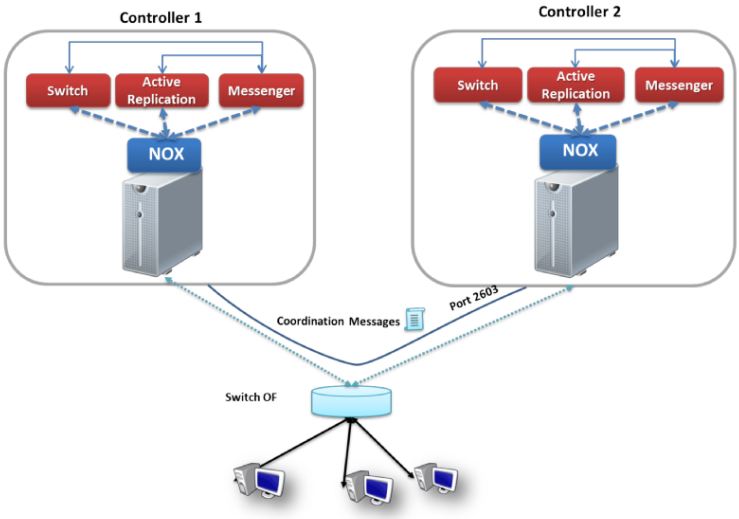


Figura 3.3: Controlador com replicação ativa[Fonseca et al. 2013].

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. Por fim, a arquitetura não escala, na perspectiva do ganho em desempenho, dado que adicionar controladores à arquitetura não aumenta o *throughput* do plano de controle.

3.3 SmartLight

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*. Neste trabalho, todos os *switches* da rede se conectam a todos os controladores disponíveis, entretanto apenas um controlador assume o papel *master*.

Para detectar a falha do controlador *master*, foi implementado um serviço de coordenação entre os controladores, de modo que em caso de falhas 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 falhas, o novo controlador principal deve recuperar os estados armazena-

dos no *data store*. A Figura 3.4 apresenta a implementação de controladores replicados passivamente por Botelho.

Nos experimentos realizados, em ambiente emulado, os autores utilizam um protótipo com dois controladores, e à medida em que variam a carga da rede e o número de *switches*, observam o comportamento do protótipo.

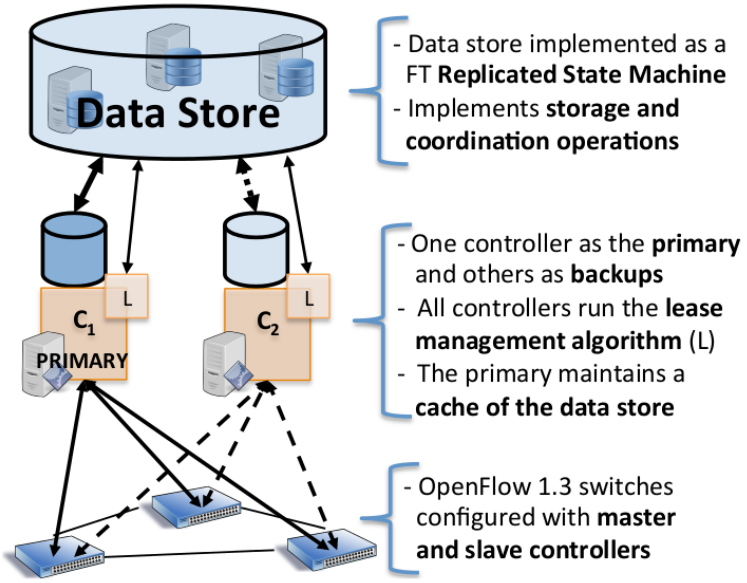


Figura 3.4: Arquitetura Smartlight[Botelho et al. 2014].

O trabalho apresenta uma linha promissora, no entanto o 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 pode adicionar uma maior latência na transição dos controladores.

Capítulo 4

Plano de Controle Resiliente

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].[1]

Por exemplo, considere uma rede de campus convencional e seus *switches*. Nesse cenário, toda a lógica de encaminhamento e o controle das configurações dos elementos de rede estão particionadas entre os próprios equipamentos. Assim, caso um desses elementos falhe, muito provavelmente parte da rede será afetada mas o restante continuará em produção. Já no caso de uma SDN implementada com o protocolo OpenFlow, esse arranjo é responsabilidade dos controladores, ou seja, toda a lógica de encaminhamento está centralizada no(s) controlador(es). Isto é, além da possibilidade de falhas nos *switches*, como no exemplo anterior, é possível que o controlador também falhe, o que, nesse caso, pode levar à inoperância da rede uma vez que novas regras de encaminhamento não seriam geradas pela duração da falha.

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

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

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 com multi-controladores, diversas perguntas ainda precisam de respostas, dentre elas destacamos as seguintes: vários controladores processam os pacotes recebidos e 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 tratadas as falhas de um controlador? Nas seções seguintes, discutiremos tais perguntas e possíveis respostas.

Em uma rede implementada com o protocolo OpenFlow, os *switches* mantêm tabelas de fluxos que ditam como o tráfego deve fluir pela rede. Quando um pacote não pode ser encaminhado, 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 fluxos 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, 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 *master* e *slave*. 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 de algumas, entre elas, mensagens de *port-status*, que informam sobre o estado das portas.

Nas próximas seções discutiremos sobre a configuração de *switches* com múltiplos controladores, fazendo o uso dos papéis OpenFlow *master*, *equal* e *slave*.

4.1 Configuração Equal

Com base nas definições dos papéis OpenFlow, podemos descrever uma configuração básica para uma rede com 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 extra, 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 4.1. 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].

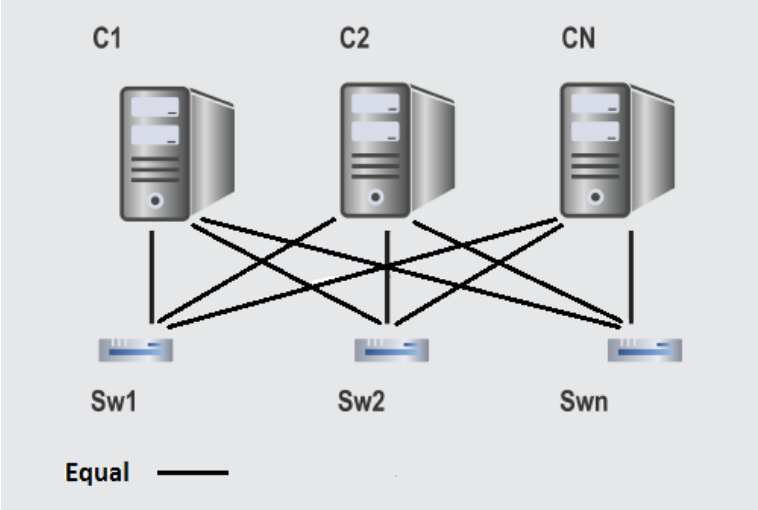


Figura 4.1: Replicação Ativa.

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

4.2 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* [Budhiraja et al. 1993], *primary-backup* ou, finalmente, *master-slave*, ilustrado na Figura 4.2, 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*.

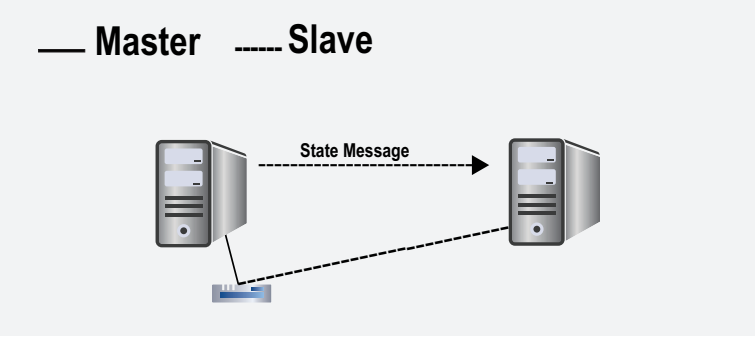


Figura 4.2: Replicação Passiva.

4.3 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?

4.4 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 redistribuídos entre os *switches*, em caso de falha dos controladores. Novamente um serviço externo pode ser usado, como por exemplo os serviços de coordenação ZooKeeper [Hunt et al. 2010] ou Doozer[2]. 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*.

4.5 Detecção de Falha

Como citado na seção anterior, em uma rede com múltiplos controladores deve existir um meio para detectar falhas que ocorram no plano de controle, de modo que a partir da detecção, ações sejam tomadas para corrigir a falha. A seguir destacamos algumas abordagens que podem ser utilizadas.

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

4.5.1 Utilizando Agentes Externos

Inicialmente, pode-se definir um agente externo para monitorar os controladores. Nessa estratégia, um serviço *Heartbeat-like* passa a verificar periodicamente se os controladores respondem às requisições enviadas pelo monitor. Caso um nó deixe de responder, o monitor detecta a falha e informa aos controladores que continuam ativos. Destacamos que como é uma arquitetura de alta-disponibilidade, e como o monitor também está sujeito a falhas, este deve possuir redundância ou algum outro mecanismo de tolerância a falhas. A Figura 4.3 ilustra o monitoramento dos controladores via um agente externo.

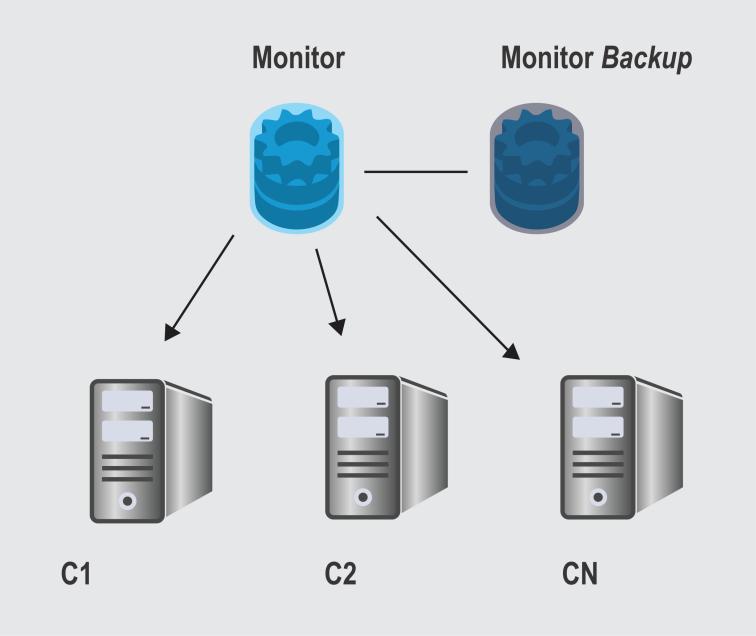


Figura 4.3: Monitoramento via agente externo.

4.5.2 Utilizando os Controladores OpenFlow

Uma outra opção é utilizar os próprios controladores da rede para detecção de falha. Ao utilizar esta estratégia, em uma rede com N controladores, cada controlador é responsável por monitorar $N - 1$ controladores. Nesse modelo o monitoramento pode ser realizado por meio de troca de mensagens *keep alive*, e em caso de falha de um controlador, todos os outros que continuam ativos detectam que o nó falhou. Essa estratégia resolve o problema apresentado seção na anterior: como existem múltiplos monitores, o mecanismo de tolerância a falhas dos monitores é inerente à implementação.

Entretanto, por existir $(N - 1) * N$ conexões de controle, pode haver um aumento substancial na troca de mensagens no plano de controle. A Figura 4.4 apresenta um dos possíveis modos de monitoramento no plano de controle.

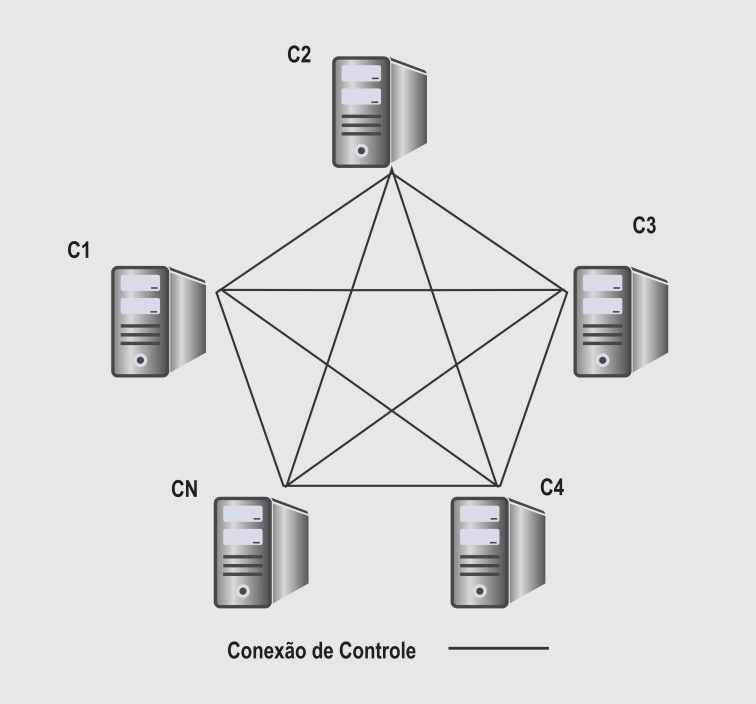


Figura 4.4: Monitoramento via controladores.

4.5.3 Utilizando os *switches* OpenFlow

Os *switches* OpenFlow armazemam uma série de dados sobre dos controladores da rede, e em um arranjo Multi-Master/Multi-Slave, este dispositivo possui informações sobre a tupla controlador/papel/*switch*. Por exemplo, em uma rede com dois controladores e dois *switches*, organizados de acordo com a Figura 4.5, o *switch* 01 mantêm uma conexão com o controlador A, e a informação de que o controlador A é o seu *master*; do mesmo modo, este *switch* mantêm um outra conexão com o controlador B, e a informação de o controlador B é o seu controlador *slave*. Caso ocorra uma falha no controlador A, considere que nenhum mecanismo de detecção/tratamento de falhas tenha sido implementado, o *switch* 01 "percebe" que o seu controlador principal caiu, e apesar de haver um outro controlador *backup*, nenhuma operação é executada para contornar a falha. Deste modo, a partir deste ponto de vista, lançamos a seguinte questão: por quê um *switch*, ao perder o seu controlador *master*, simplesmente não sinaliza a um dos $N-1$ controladores da rede para que um novo *master* assuma o processamento de seus pacotes?

A partir dessas considerações, seguimos na direção de que o *switch* com todas as informações sobre os controladores da rede e ainda pelo fato de manter conexões com estes, tem a possibilidade de empreender um novo mecanismo de detecção de falhas em controladores SDN. Nesta nova estratégia, os *switches* passam a monitorar o seu controlador *master*, e em caso de falha, enviam uma mensagem que todos os controladores possam processar. Então, um controlador deve ser escolhido para assumir este *switch* "órfão".

Essa estratégia apresenta algumas vantagens sobre outros modos de monitoramento.

Como cada *switch* monitora apenas o seu controlador *master*, ou seja, a relação é de 1:1, a complexidade de monitorar múltiplos controladores é distribuída entre os *switches* da rede. Um outro ponto a ser destacado é o fato do *switch* manter informações e conexões de/com todos os controladores da rede, o que faz esse estratégia ser pouco invasiva. Por fim, esse mecanismo de detecção permite que aplicações e usuários que não necessitem manter a consistência entre os controladores, possam usufruir de um ambiente multi-controlador de modo muito simples. A Figura 4.5 ilustra o monitoramento via o plano de dados.

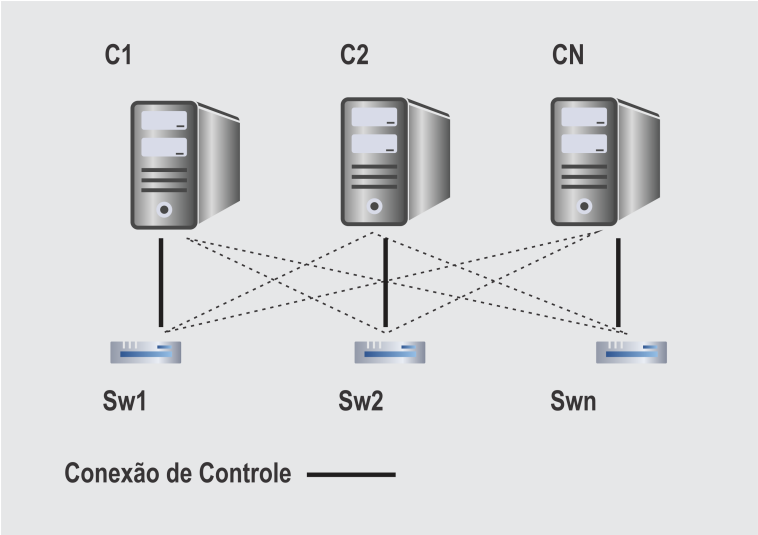


Figura 4.5: Monitoramento via *switches*.

Em todas as estratégias citadas anteriomente, mais de um controlador pode ser notificado sobre a falha de controladores, assim, deve haver um mecanismo para que os controladores entrem em consenso sobre qual controlador será escolhido para assumir o *switch* que perdeu o seu controlador *master*. Dentre diversas soluções, para o contexto discutido, descrevemos algumas opções:

1. O monitor mantém uma lista de prioridades, e a partir dessa lista define qual controlador será solicitado para assumir o *switch* "órfão";

2. Os controladores mantêm uma lista de prioridades, onde é definida a sequência dos controladores que assumem em caso de falha, e após receber a sinalização do monitor, o controlador com a prioridade mais alta assume;

3. O monitor, de modo aleatório, define qual controlador será solicitado para assumir o *switch* "órfão";

4.6 Considerações parciais

Algumas considerações podem ser feitas em relação à 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. À medida que controladores são adicionados à 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, a coordenação das réplicas, e detecção e tratamento de falhas. 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.

No próximo Capítulo contextualiza-se a arquitetura proposta, onde detalhamos a implementação do protótipo e suas principais características.

Capítulo 5

Implementação do Protótipo

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

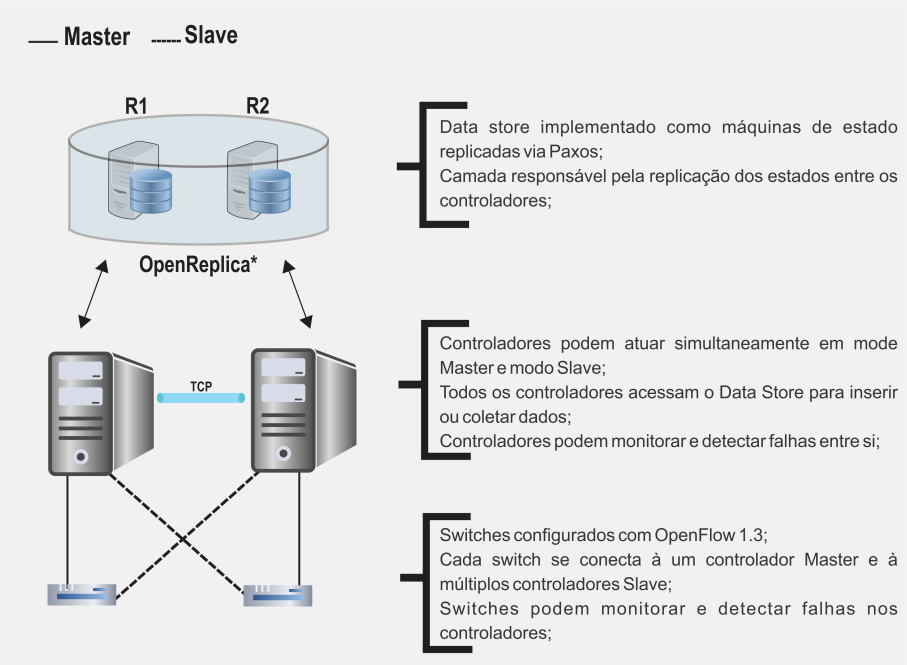


Figura 5.1: Proposta de arquitetura Resiliência para o plano de controle OpenFlow.

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. Na implementação do protótipo é considerado um modelo de sistema síncrono com falhas de controladores, do tipo *fail-stop*; os canais de comunicação não duplicam nem perdem mensagens.

Como consequência do atendimento às premissas arquiteturais, segue-se com o de-

senvolvimento do trabalho a partir de três perspectivas: a distribuição do estado entre os controladores, o plano de controle, e o plano de dados. Deste modo, é abordado os desafios e as soluções utilizadas para o *design* do protótipo implementado, ilustrado na Figura 5.1.

Nas próximas subseções serão descritas as abordagens utilizadas nas soluções para implementação do protótipo.

5.1 Data Store

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. Sendo assim, é oportuno detalhar a solução escolhida para manter os estados dos controladores consistentes.

Por exemplo, considere uma rede com diversos controladores executando uma aplicação de *switching*: à medida que em que os pacotes fluem pela rede, são processados pelos controladores, o controlador armazena os endereços MAC dos *hosts*, e qual porta de qual *switch* esses endereços estão associados. Caso haja uma falha em um dos controladores da rede, todo o mapeamento que estava armazenado no controlador que falhou deve estar disponível para o restante dos controladores da rede.

Nossa proposta utiliza um serviço aberto chamado OpenReplica[1] para manter a consistência do estado da rede. Este 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 5.2).

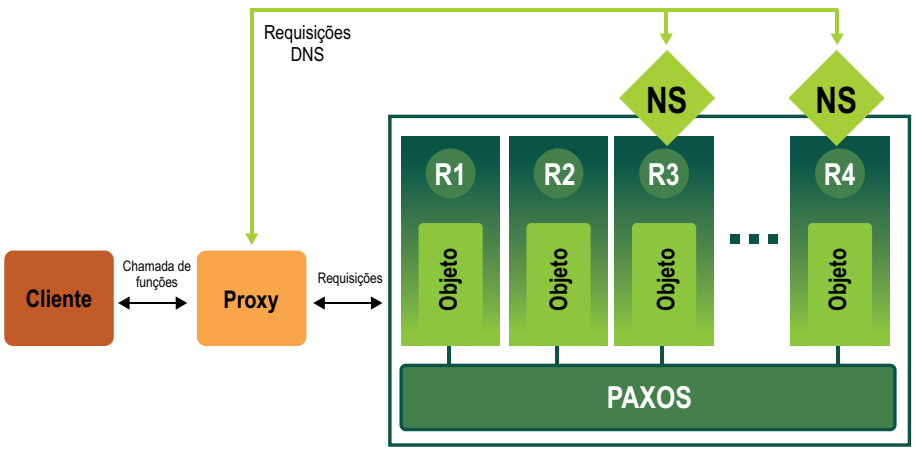


Figura 5.2: Open Replica para replicação e sincronização do estado.

[1] http://openreplica.org/

Na implementação do protótipo, define-se uma estrutura de dados chamada de NIB (Network Information Base), que pelo fato da abordagem do serviço de replicação ser OO (Orientado a Objetos), também pode ser chamada de objeto NIB. Nessa estrutura são armazenadas as informações compartilhadas entre os controladores, como por exemplo, os mapeamentos controlador/papel/*switch* e endereço MAC/porta/*switch*. Assim, o objeto NIB acessado via OpenReplica reflete a visão global da rede SDN.

Como característica de programação OO, os atributos dos objetos devem ser acessados por meio de métodos, e seguindo este paradigma, o objeto NIB possui um conjunto de métodos para inserir, alterar e excluir dados de sua estrutura. Esse métodos são executados pelos controladores. De um modo simplificado, a abstração provida pelo OpenReplica pode ser vista como uma porção de memória virtual comum a todos os controladores da rede.

Uma outra característica inerente à escolha deste serviço de replicação diz respeito à fácil integração com o controlador Ryu. Pelo fato de ambos serem escritos em Python é dispensável o uso de *binds*, o que torna a integração entre os sistemas mais simples.

5.2 Plano de Controle

O sistema operacional de redes escolhido para atuar como controlador utilizado no protótipo foi o Ryu, versão 3.18, em virtude deste ser um ambiente com suporte completo às versões 1.0, 1.2, 1.3 e 1.4 do protocolo OpenFlow. Outros pontos importantes para esta escolha foram os fatos de se tratar de um projeto de código aberto; ter suporte de uma comunidade ativa, o que faz o controlador estar alinhado com a evolução do OF; e dispor de farta documentação oficial, o que inclui um *e-book* [Team 2013].

O controlador Ryu não dispõe de nenhuma *feature* para atuar em conjunto com outros controladores, deste modo, todo mecanismo de comunicação entre controladores e/ou *data store* fica a cargo das implementações do projetista de redes. No modelo sugerido, pode haver múltiplos controladores na rede, e a princípio deve ser permitida a comunicação entre estes.

Em cada controlador as funções do sistema são divididas em duas aplicações que são executadas em paralelo:*ConnectionManager* e *LearningSwitch*. A *ConnectionManager* é responsável por manter as principais funções do protótipo, dentre elas, realizar a comunicação com o *data store*, com outros controladores da rede e com a aplicação de *switching*; detectar e monitorar falhas nos controladores; e gerenciar a conexão com os *switches*, já que o controlador pode assumir diferentes papéis para diferentes *switches*. Já a segunda aplicação executa uma lógica de um *Learning Switch* adaptada para receber eventos do Ryu. Figura 5.3 ilustra a estrutura organizacional das aplicações *ConnectionManager* e *LearningSwitch*.

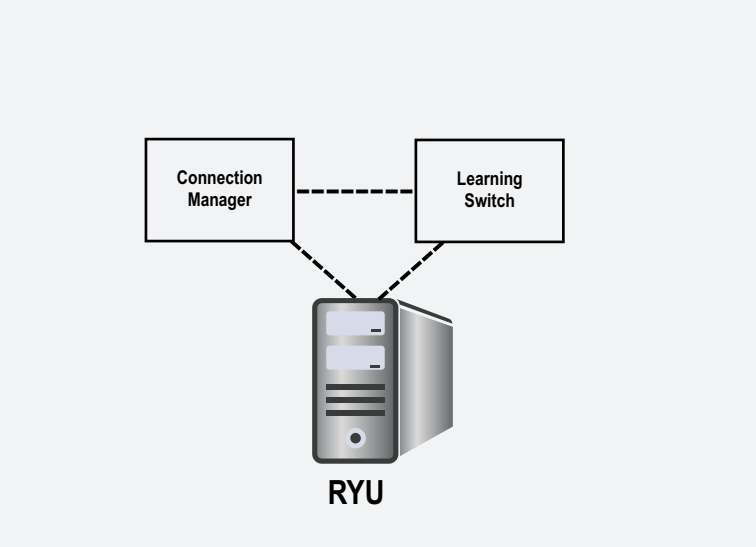


Figura 5.3: Organização do controlador implementado.

5.2.1 Detecção de Falhas via Plano de Controle

Como descrito na seção anterior, nessa estratégia multi-controlador podemos ter N controladores processando pacotes do plano de dados, e todos os controladores deste arranjo estão sujeitos a falhas. Isto é, em uma rede com múltiplos controladores, caso um deles deixe de funcionar, deve existir um mecanismo que detecte essa falha e sinalize para os controladores que continuam operacionais, a falha de um nó da rede. Para tanto, inicialmente, é definido que os próprios controladores podem ser responsáveis pela detecção das falhas do plano de controle.

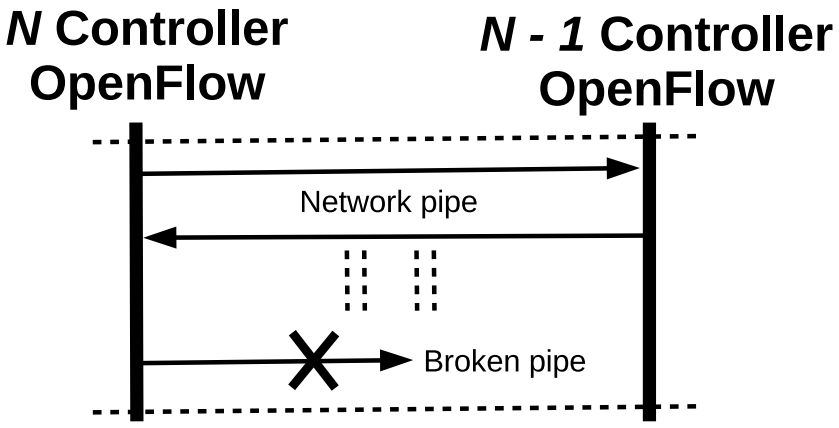


Figura 5.4: Detecção de falha via controladores OpenFlow.

Na implementação do protótipo apresentado, no caso de uma rede com N controladores, cada controlador deve abrir uma conexão com $N-1$ controladores. Deste modo, cada controlador utiliza mensagens do tipo *keep-alive*, enviadas por estas conexões, para monitorar o restante dos controladores da rede. Como as aplicações do controlador Ryu são escritas em Python, utiliza-se a interface BSD *socket* da linguagem para implementar

tais conexões, e caso ocorra uma falha que torne um dos controladores indisponíveis, $N-1$ controladores detectam a falha. A Figura 5.4 apresenta a técnica implementada no plano de controle para detectar falhas nos controladores.

Ao receber a notificação de falha, os múltiplos controladores devem entrar em consenso sobre qual controlador irá assumir a propriedade do *switch* "órfão". O mecanismo implementado para determinar qual controlador irá assumir é baseado em uma lista de prioridades armazenada no *data store*, sendo assim, a lista é de comum acesso para todos os controladores da rede.

5.2.2 Tratamento da Falha

Uma vez detectada a falha em um dos controladores, um procedimento deve ser executado a fim de contornar o problema causado pela indisponibilidade de tal controlador, com o objetivo de minimizar o impacto no funcionamento da rede. Essa rotina foi definida como **Recuperação de Falha**.

A recuperação de falha consiste em um processo que faz com que um dos controladores, que continua ativo, torne-se o novo *master* para todos os *switches* que perderam o seu controlador principal, isto é, o seu controlador *master*. Então, os passos executados na operação, são: no momento em que a falha é detectada pelos controladores, uma função da aplicação *ConnectionManager* acessa os dados da NIB no *data store* e verifica se o seu ID é o designado para assumir as funções do controlador que falhou. Caso seja verdadeiro, este controlador 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 operação. A Figura 5.5 ilustra a comunicação do controlador com o *switch* "orfão".

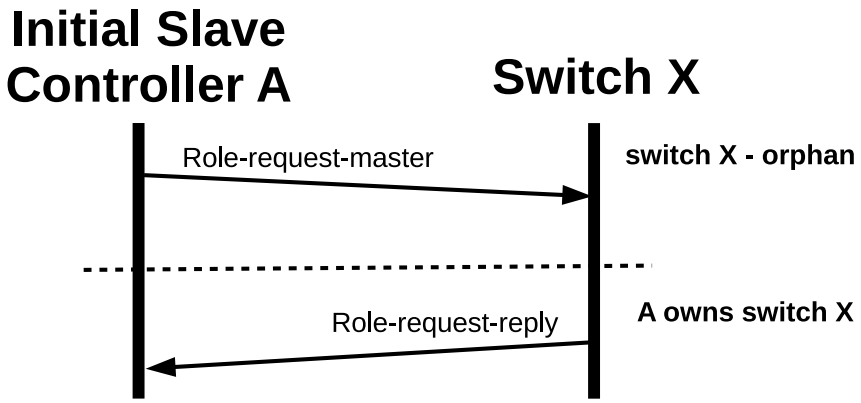


Figura 5.5: Processo de Recuperação de Falha.

Em [Dixit et al. 2013] os autores apresentam um protocolo de migração de *switches*. Nesta técnica, com o uso de mensagens OpenFlow é possível migrar *switches "on the fly"* entre controladores. A partir de uma variação da técnica apresentada por Dixit, adicio-

namos uma outra operação a ser utilizada no protótipo do plano de controle resiliente, a **Restauração da Falha**. Essa ação ocorre quando o controlador que falhou está disponível novamente, e reassume os *switches* que o tinham como *master* no momento anterior à falha.

Inicialmente, pode-se sugerir que um controlador que volta de uma falha e deseja reassumir os seus *switches*, deve simplesmente enviar uma mensagem de *change-role-request* solicitando que seu papel seja novamente *master* para os *switches* em questão. Todavia, quando um *switch* recebe uma solicitação para responder a um novo controlador *master*, este *switch* automaticamente reduz o seu controlador, que era *master* até então, para o papel *slave*. Assim, caso o controlador *master* inicial possua respostas pendentes para um *switch* X e passe, de modo abrupto, para o modo *slave*, a este não será permitido o envio de fluxos pendentes, já que controladores com papel *slave* não têm permissão de escrita na tabela de fluxos de *switches*. Portanto, um mecanismo que permita uma transição suave de *switches* entre controladores é necessário.

No processo de restauração de falha, o controlador que volta de um período de indisponibilidade, no momento da inicialização de suas aplicações, acessa o *data store*, e percebe que está voltando de uma falha. A partir então, assume o papel *equal* para todos os *switches* que o tinham como master, entretanto não processa *packet-in* algum. Esse procedimento otimiza a execução da migração, já que as mensagens de troca de papel são enviadas na inicialização do controlador. Após essa operação, considerando o controlador A, o controlador B (voltando de uma falha), e um *switch* X, os passos executados no protocolo de migração são definidos em quatro fases:

- Fase 1: o controlador B envia uma mensagem de migração *start-migration* para o controlador A, inicializando o protocolo;

- Fase 2: o controlador A envia uma mensagem do tipo *flow-mod* ao *switch* X, adicionando um *dummy-flow*; logo em seguida, A envia uma outra mensagem do tipo *barrier-request*. Essa mensagem garante que todos pacotes que tenham chegado anteriormente à *barrier-request* sejam processados antes que qualquer pacote que chegue após a mensagem de barreira; o *switch* responde com uma mensagem *barrier-reply*; então, o controlador A envia uma mensagem *flow-mod* ao *switch* X excluindo o *dummy-flow* adicionado alguns passos atrás; o *switch* X responde com uma mensagem *flow-removed*, que tanto o controlador A quanto o B recebem. Essa mensagem funciona como um gatilho para o controlador B, indicando que ele assumirá o controle do *switch* em questão;

- Fase 3: como podem existir pacotes pendentes para ainda serem processados por A, uma nova mensagem de barreira é enviada ao *switch*; assim que o controlador A recebe a *barrier-reply*, A envia uma mensagem ao controlador B finalizando a migração;

- Fase 4: B por sua vez, envia uma mensagem *role-request master* para X, que então responde com uma mensagem *role-reply*. A Figura 5.6 apresenta os passos para a migração de *switches*.

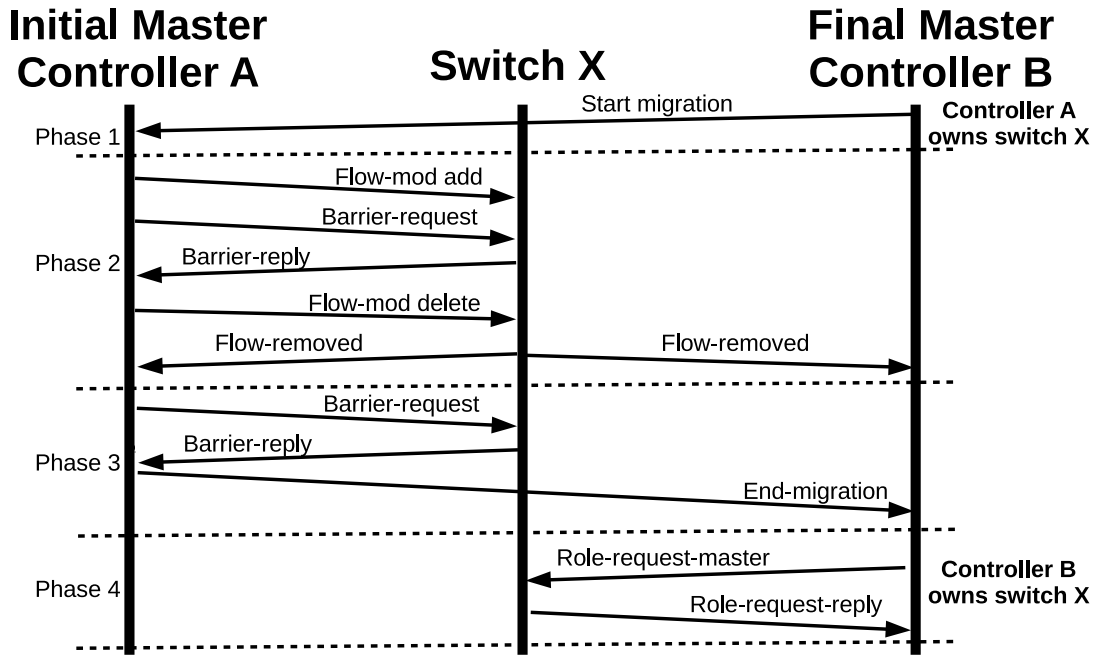


Figura 5.6: Processo de Restauração da Falha.

Por fim, cabe a aplicação de *ConnectionManager* definir os papéis de cada controlador em relação aos *switches* da rede. De acordo com as definições do OpenFlow, para cada *switch* deve haver um, e apenas, um controlador *master*. No protótipo implementado os *switches* são configurados de modo alternado, isto é, o *switch* 1 tem o controlador A como *master* e o controlador B como *slave*. Já o *switch* 2 tem o controlador B como *master* e o controlador A como *slave*.

5.3 Plano de Dados

Existem diversos equipamentos com suporte OpenFlow disponíveis no mercado, entretanto, em geral, esses equipamentos apresentam diversas limitações, e talvez a principal seja o fato de serem implementações fechadas. Um outro ponto é o desinteresse dos grandes fabricantes em acompanhar a evolução do protocolo, o que leva a opções de *switches* com versões mais antigas do protocolo OpenFlow ou com implementações parciais, isto é, apenas algumas funções estão disponíveis.

A partir de um projeto desenvolvido no NERDS, grupo de pesquisa ligado à UFES, foi criada uma plataforma aberta para experimentação em *switches* OpenFlow.

Nesse projeto utilizamos equipamentos da Mikrotik, uma empresa da Letônia fundada em 1995, responsável pelo desenvolvimento do sistema operacional RouterOS, e posteri-

ormente pela construção do *hardware* embarcado Mikrotik RouterBoard. Combinados, eles são destinados ao mercado de pequenos a médios provedores de serviço Internet. O Mikrotik RouterOS é o sistema operacional embarcado originalmente no *hardware* RouterBoard, podendo também ser instalado em computadores com arquitetura x86. Baseado no *Kernel* do Linux versão 2.6, traz consigo uma gama de recursos, como roteamento, *firewall*, controle de banda, ponto de acesso sem fio, *hotspot*, dentre outros [Mikrotik 2014].

A partir de sua versão 6.0, o RouterOS possibilita a utilização do protocolo OpenFlow versão 1.0, através da instalação de seu respectivo pacote. Por ser um *software* proprietário de plataforma fechada, o RouterOS não permite modificações e atualizações além do OpenFlow 1.0 padrão, disponibilizado pelo fabricante.

O projeto da plataforma aberta iniciou-se pela substituição do sistema operacional, originalmente RouterOS que suporta apenas OF 1.0 (Plataforma Fechada), pelo OpenWRT [OpenWRT 2014] que permite a instalação dos *switches* virtuais (OvS e Softswitch13) que suportam novas versões do protocolo OF (por exemplo a versão 1.3).

Em nossos experimentos utilizamos o OpenWRT, que é uma distribuição GNU/Linux altamente extensível para dispositivos embarcados, dentre eles, o Routerboard utilizado neste trabalho. Diferentemente dos demais *softwares* originais desses equipamentos, o OpenWRT foi construído a partir do zero para ser um sistema operacional completo e facilmente modificado, utilizando um *kernel* do Linux mais recente que a maioria das outras distribuições. Como máquina de encaminhamento, utilizamos o Open vSwitch (OvS) que é um *switch* virtual que segue a arquitetura OpenFlow, também é implementado em *software*, mas com o plano de dados implementado diretamente no *kernel* do Linux, enquanto funções de controle são implementadas em modo de usuário [Pfaff et al. 2009, Subramanian et al. 2009].

5.3.1 Detecção de Falhas via Plano de Dados

Nesta seção é proposta uma alternativa mais simples e eficaz de implementar a detecção da queda de um controlador SDN. O objetivo é compartilhar esta tarefa com o plano de dados, mas sem alterar as funções que são específicas de cada plano.

Sabe-se que um *software switch*, tal como o OvS, armazena muitas informações sobre os controladores SDN (por exemplo, IP, porta e papel) na base de dados "ovsdb". Além disso, na arquitetura escolhida, multi-*master*/multi-*slave*, os *switches* mantêm uma conexão SSL ativa com cada controlador da rede. Essas informações permitem que um novo mecanismo de detecção de falhas, implementado no plano de dados, possa ser definido de modo simplificado, já que inúmeras informações sobre os estados dos controladores podem ser monitorados a partir dos *switches* OpenFlow. Deste modo, é apresentada uma nova estratégia para detecção de falhas: estender o daemon do OvS "vswitchd" incluindo uma nova funcionalidade, que passará a monitorar o seu controlador *master*, e em caso

de falha, poderá enviar uma mensagem para todos os controladores SDN ativos.

Existe uma restrição muito grande sobre os tipos de pacotes que controladores em modo *slave* podem processar, entretanto, mensagens do tipo *port-status* são processadas por todos os controladores que o *switch* estiver conectado. Assim, o *switch* ao detectar uma falha em seu controlador *master*, gera uma mensagem *port-status* que será recebida por todos os controladores da rede. A Figura 5.7 ilustra o novo protocolo proposto, que está definido nas seguintes fases:

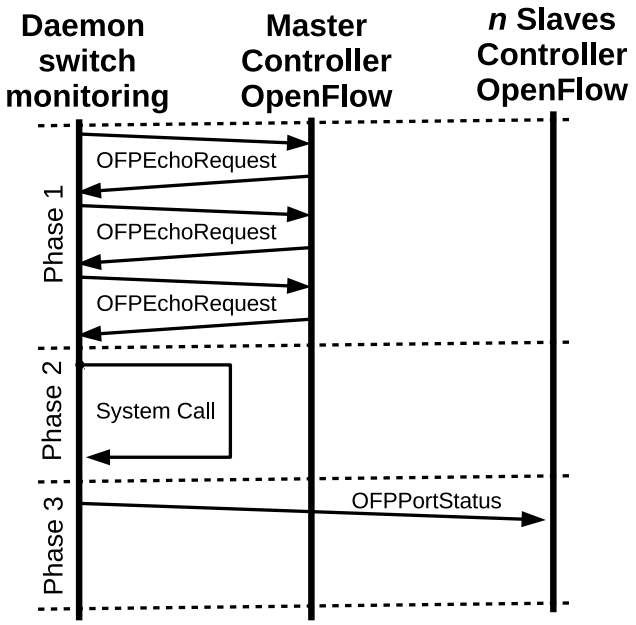


Figura 5.7: Detecção de falhas via *switch* OpenFlow.

- Fase 1: trata do monitoramento, este será realizado através da troca de mensagens utilizando o canal de comunicação já existente entre o *switch* e o controlador *master*. Adotamos como modelo de referência a troca de mensagens síncronas *EchoRequest* em intervalos de 3.33 ms (mensagens CCMs) [Forum 2013].

- Fase 2: se o controlador *master* não responder após três tentativas seguidas, o daemon executará uma chamada de sistema que irá adicionar uma porta virtual no switch. Nesta fase consideramos que há uma falha na comunicação lógica/física entre switch e controlador. Conforme especificação do OpenFlow, qualquer *add/delete/modify* de porta, seja física ou virtual, irá gerar um evento. Este evento é chamado *PortStatus* e será recebido por todos os controladores, cujo *switch* esteja conectado.

Os controladores *slaves* associados ao *switch*, assim que recebem a mensagem *PortStatus* executam a política programada para assumir este switch "órfão".

Capítulo 6

Avaliação Experimental

6.1 Plataforma de Avaliação

Na avaliação experimental foram definidos dois ambientes para análise do protótipo: ambiente real e ambiente emulado.

No ambiente real foram utilizados *switches* comerciais de prateleira (COTS), que foram 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 as RouterBoards, é possível inferir como o protótipo implementado se comportar em situações reais.

Os equipamentos utilizados para realizar os experimentos no ambiente real, foram: 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).

Já no ambiente emulado, utiliza-se a ferramenta Mininet[Handigol et al. 2012]. Neste caso, o equipamento utilizado para os experimentos no ambiente emulado, foi: um computador (Core i5 3.2GHz, 6GB de RAM, HD 320GB 5400rpm, SO Ubuntu 14.04 64bits).

6.1.1 Metodologia

Para permitir o mínimo de interferência entre os cenários, nos dois ambientes, as ferramentas foram reinicializadas 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:

- Ambiente Real: latência de recuperação e restauração de falha dos controladores, com diferentes taxas de tráfego no plano de controle; latência de recuperação e

restauração de falha dos controladores, com diferentes taxas de tráfego no plano de dados; latência de *packet-in* nos controladores, antes e após a ocorrência falhas; latência na detecção de falhas no plano de dados; latência na detecção de falhas no plano de controle; e o consumo de processador das RouterBoards nos diferentes cenários apresentados.

- Ambiente Emulado: razão da Replicação x Desempenho do sistema, à medida em que varia-se a quantidade de controladores e a carga do plano de controle;

6.1.2 Carga de trabalho

De modo a gerar o tráfego necessário de acordo com as demandas de cada cenário nos ambientes citados, foi criado um gerador de tráfego em Perl que utiliza a extensão NET::RawIP para criar pacotes IP a uma taxa predefinida. No ambiente de experimentação, os controladores não foram o gargalo dos testes. Ao contrário, por conta do *hardware* utilizado, seria possível aumentar a taxa de envio de pacotes. Já no plano de dados, devida à limitação do *hardware* utilizado, em especial da *CPU*, limitamos a taxa de geração de pacotes a no máximo 1.000 *packet-in/s*. Então, as taxas de envio predefinidas foram: 250, 500, 750 e 1.000 *packet-in/s*. Assim, mesmo no ambiente emulado, tomamos como base o limite imposto pelos *switches* utilizados no ambiente real.

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[1]. 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.

6.1.3 Medições

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 mostrou-se aceitável para garantir a confiabilidade do resultado as experimentações.

Para medir as operações realizadas pelos controladores, foram inseridos quatro *timestamps* no código. Na recuperação de falha, assim que o controlador inicia o processo de recuperação de falhas, 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

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

voltou da falha recebe a mensagem *change-role-reply*, que sinaliza o término da operação.

Na medição da latência dos pacotes nos controladores, o tráfego foi capturado por meio da ferramenta *tshark*, e posteriormente analisada com o *wireshark*[2], de modo que cada *packet-in* tivesse o tempo medido de acordo com o seu *packet-out*.

Finalmente, para medir o percentual de uso de CPU das RouterBoards, a ferramenta de monitoramento *mpstat*[3] foi utilizada.

Todos os testes/coletas de dados foram sistematizados por meio de *script*, e posteriormente armazenados em arquivos. Os seguinte passos foram executados para a coleta de cada amostra:

1. inicialização do controlador 1;
2. inicialização do controlador 2;
3. inicialização do tshark (para testes com foco no plano de controle);
4. inicialização do mpstat;
5. inicialização do gerador de pacotes (testes com foco no plano de controle) ou inicialização do iperf (para testes com foco no plano de dados);
6. desligamento do controlador 2 (simulação de falha);
7. reinicialização do controlador 2 (simulação de restauração de falha);
8. finalização do gerador de pacotes(para testes com foco no plano de controle) ou finalização do iperf (para testes com foco no plano de dados);
9. finalização do mpstat;
10. finalização do tshark (para testes com foco no plano de controle);
11. finalização do controlador 1;
12. finalização do controlador 2;

Para os experimentos no ambiente emulado, os seguintes passos foram seguidos:

1. inicialização dos controladores;
2. inicialização do tshark;
3. inicialização do gerador de pacotes;

[2]https://www.wireshark.org/
[3]http://www.linuxcommand.org/man_pages/mpstat1.html

4. finalização do gerador de pacotes (para testes com foco no plano de controle) ou finalização do iperf (para testes com foco no plano de dados);

5. finalização do tshark;

6. finalização dos controladores;

6.2 Discussão dos Resultados

Nesta seção discute-se os resultados obtidos. Estes foram organizados obedecendo a seguinte sequência: primeiro, são apresentados os resultados dos experimentos realizados no ambiente real. Em seguida, os resultados do ambiente emulado; e por fim, alguns comentários sobre os resultados.

Os valores de 1.000 *packet-in/s* e 720 *Mbps* representam o limite de processamento dos *switches* utilizados, isto é, não recomenda-se a sua utilização acima desta carga de 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 foram apresentadas graficamente, no entanto, como são resultados relevantes foram comentados no texto.

6.2.1 Avaliação do Ambiente Real

6.2.1.1 Desempenho do plano de controle

Nas medições realizadas para o processo de recuperação de falha, ilustradas na Figura 6.1, 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 6.1, 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), mas com alta variabilidade.

Os resultados acima não consideram o tempo de detecção da falha, apenas os tempos de recuperação e restauração da falhas são apresentados. De todo modo, o tempo do processo de restauração da falhas é maior que o de recuperação, haja visto que a restauração possui maior troca de mensagens entre o controlador e o *switch*. A partir da taxa de 750 *packet-in/s* os tempos tendem a aumentar consideravelmente, chegando ao ápice com a taxa de

1.000 *packet-in/s*. Esse comportamento pode ser justificado pela alta taxa de ocupação de CPU dos *switches* no referido experimento.

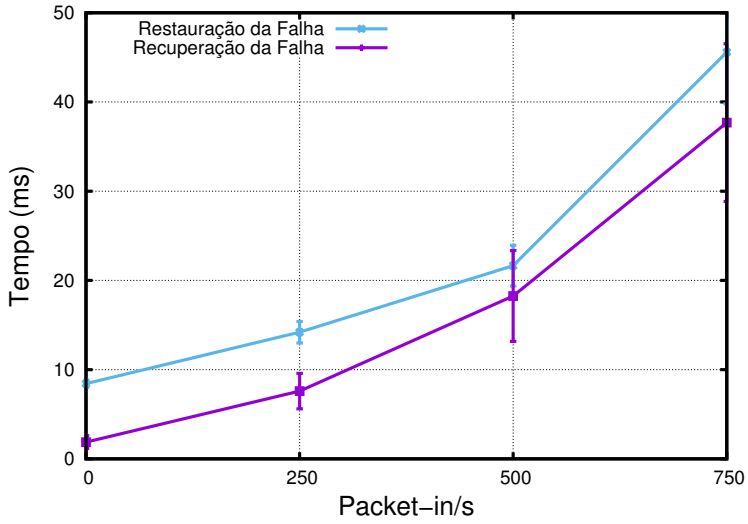


Figura 6.1: Recuperação e Restauração de Falha - Tráfego no Plano de Controle

Para a carga de processamento das RouterBoards, os resultados apresentados na Figura 6.2, de acordo com as taxas de geração de pacotes, variaram entre 77% e 99%. Com o envio a 250 *packet-in/s*, obteve-se o menor percentual de uso de processador, 77%; ao subir a taxa para 500 *packet-in/s*, o uso foi a 84%; no terceiro experimento, a taxa de 750 *packet-in/s*, chegou a 96%; e por fim, quando a taxa de envio era 1.000 *packet-in/s*, alcançou 99% de uso, número próximo à ocupação total do processador.

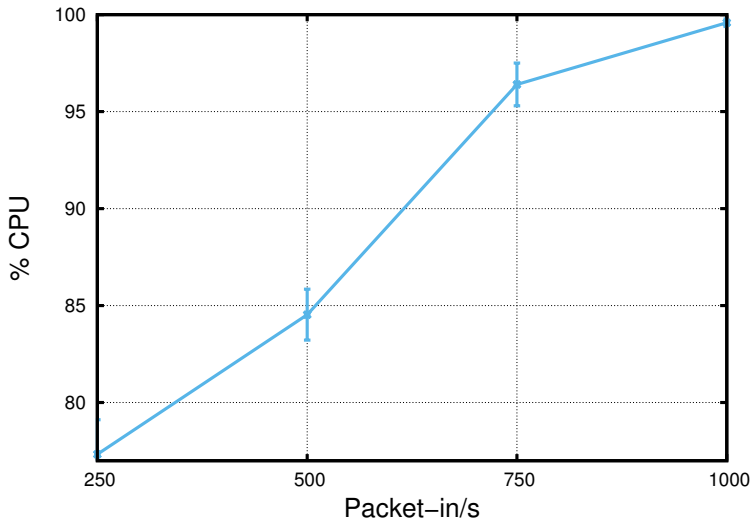


Figura 6.2: Carga de Processamento nas RouterBoards - Tráfego no Plano de Controle

Desde a menor taxa de envio de pacotes, 250 *packet-in/s*, o percentual de uso da CPU dos *switches* foi alta. Em outras palavras, isso significa que um *switch* OpenFlow necessita de muitos ciclos de CPU para enviar pacotes para o plano de controle, o que

pode ser confirmado pelo pequeno intervalo em que o uso da CPU variou, com números altos, entre 77 e 99%.

Em relação às latências do processo de detecção de falhas quando executado no plano de controle, isto é, quando os próprios controladores monitoram-se, os resultados são: em um primeiro momento, sem tráfego na a rede, a latência para a detecção da falha foi 69,43 ms; com o início do envio de pacotes, com 250 *packet-in/s* subiu para 71,44 ms; quando a taxa subiu para 500 *packet-in/s*, a latência foi a 72,25 ms; com a taxa de 750 *packet-in/s* o resultado foi ligeiramente maior que o anterior, alcaçando 72,58; por fim, quando a taxa de envio era de 1.000 *packet-in/s*, a latência subiu para 72,98 ms.

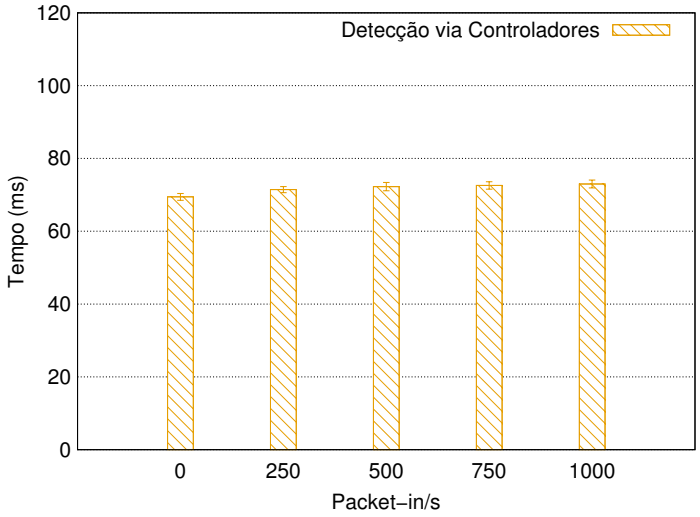


Figura 6.3: Latência de Detecção de Falhas pelos Controladores

A Figura 6.3 apresenta os resultados para o tempo de detecção de falha executada pelos controladores. Nesta série de experimentos os resultados seguiram com baixa variação, já que os controladores não foram submetidos a sobrecarga.

Em nossos experimentos também avaliamos a latência do processamento de pacotes pelos controladores. O objetivo foi avaliar o impacto na latência do processamento de *packet-in* quando um controlador falha e um outro assume o seu papel. Por exemplo, em um dos cenários apresentados, a taxa de envio é 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 e 8.200 μs. Os valores para a latência do processamento de *packet-in* são apresentados na Figura 6.4.

Em geral, a latência para o processamento de pacotes cresceu gradativamente à medida em que a taxa de envio aumentou, e na média dobrou após a falha de um controlador. Entretanto, no último experimento, quando a taxa de envio era de 1.000 *packet-in/s*, após a falha, a latência foi oito vezes maior. Este valor pode ser explicado pela alta taxa de

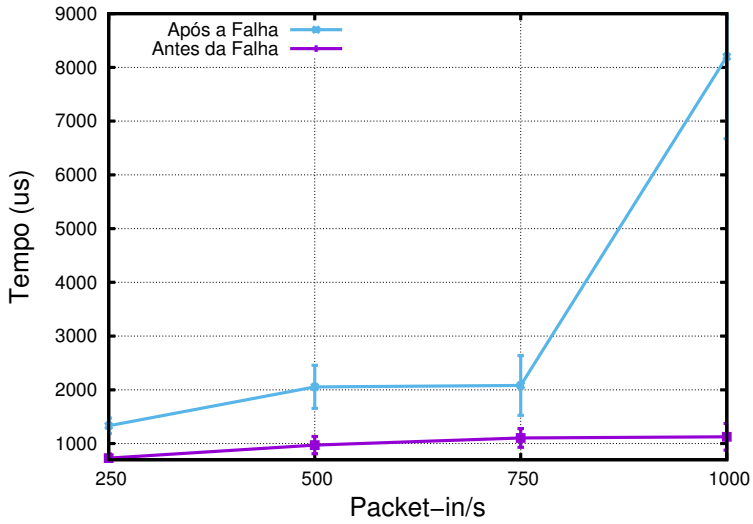


Figura 6.4: Latência Processamento de Packet-in - Tráfego no Plano de Controle

pacotes que o controlador tem de processar e pelo consumo da CPU dos *switches* estar próximo de 100% de sua capacidade.

6.2.1.2 Desempenho do plano de dados

Após os resultados dos experimentos com tráfego no plano de controle, é apresentado os dados dos experimentos com variação de carga no plano de dados. O objetivo desta avaliação foi analisar o impacto que tráfego no plano de dados (encaminhamento dos pacotes) representa na latência para as operações do protótipo. Os resultados segue a mesma ordem dos experimentos apresentados anteriormente, e são mostrados na Figura 6.5.

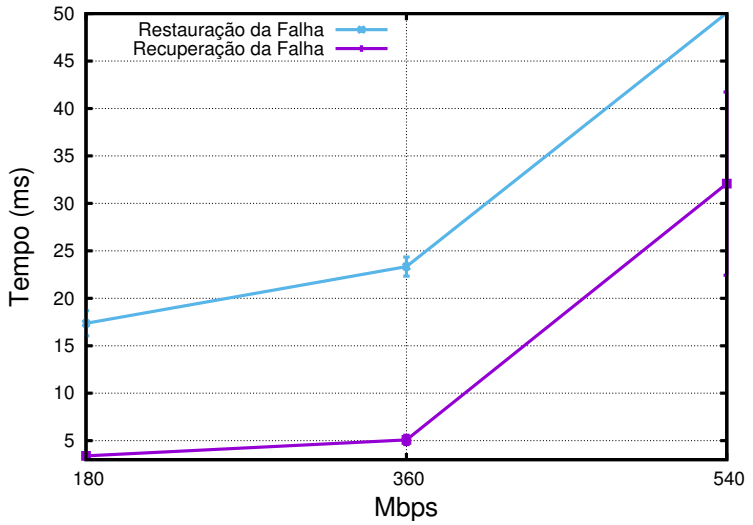


Figura 6.5: Recuperação e Restauração da Falha - Tráfego no Plano de Dados.

O primeiro resultado refere-se à 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 55,51 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 55,51 ms.

No processo de restauração de falha, os resultados apresentados estão ilustrados na Figura 6.5: com taxa de 180 Mbps, a latência ficou em 17,37 ms; a 360 Mbps, subiu para 23,34 ms; com o aumento do tráfego para 540 Mbps, alcançou 50,06 ms; e ao final, com taxa de 720 Mbps, chegou a 119,28ms.

Seguindo a tendência dos resultados discutidos nos experimentos realizados com carga no plano de dados, o processo de recuperação de falha continua levando menos tempo que a restauração. Enquanto os *switches* não estão sobrecarregados, os valores são incrementados linearmente, já a partir da taxa de 540 Mbps há um claro aumento no tempo para execução dos processos.

A seguir, na próxima série de experimentos realizados no ambiente real, exibimos os dados sobre a utilização de CPU das RouterBoards, Figura 6.6. Diferentemente do cenário em que a carga era enviada ao plano de controle, encaminhar pacotes no plano de dados mostrou-se uma operação menos dispendiosa. Para os experimentos realizados o percentual de utilização variou entre 26% e 95%, e seguiu um comportamento de crescimento linear.

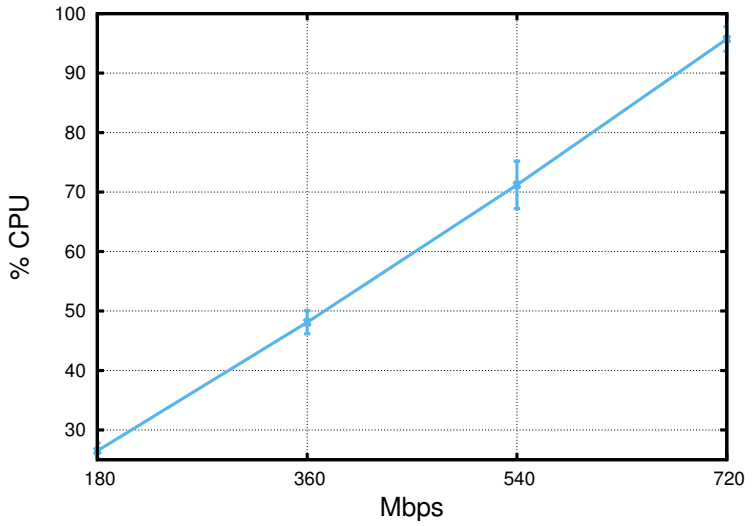


Figura 6.6: Carga de Processamento nas RouterBoards - Tráfego no Plano de Dados

Nas medições do processo de detecção de falha quando executada pelos *switches*, ou seja, no plano de dados, os valores da latência variaram entre 32,23 e 259,05 ms. Neste experimento os valores das latências aumentaram de modo gradual até a taxa de 750 *packet-in/s*, no último experimento, com 1.000 *packet-in/s* a latência foi aproximadamente oito vezes maior que o menor valor apresentado. O valor elástico nesta última medição deve-se ao fato de que com esta taxa de envio de pacotes o processador do *switch* estar

próximo de sua capacidade máxima, conforme ilustrado na Figura 6.7.

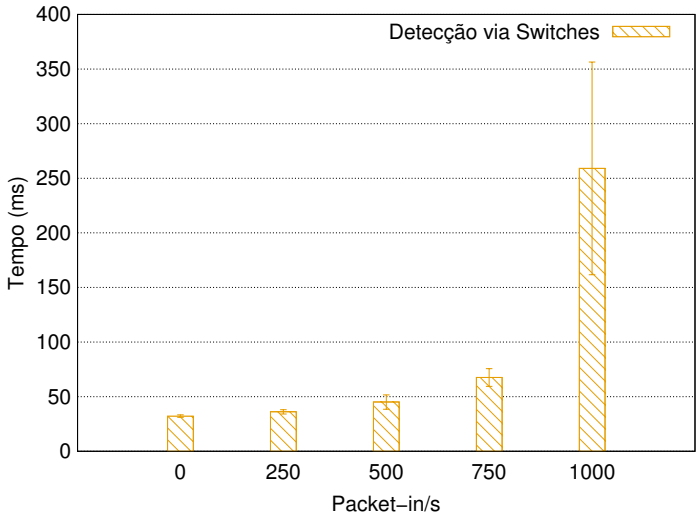


Figura 6.7: Latência de Detecção de Falhas pelos *Switches*.

A seguir avaliamos o impacto do componente de detecção de falha no processo completo para recuperação de falha, já que até então a detecção não havia sido considerada. A recuperação de falha completa compreende a tempo de detecção de falha somado ao tempo de recuperação de falha. Também foi realizado um comparativo da detecção de falha executada nos controladores *versus* a detecção de falha executada nos *switches*. A Figura 6.8 exibe os valores agrupados destas medições.

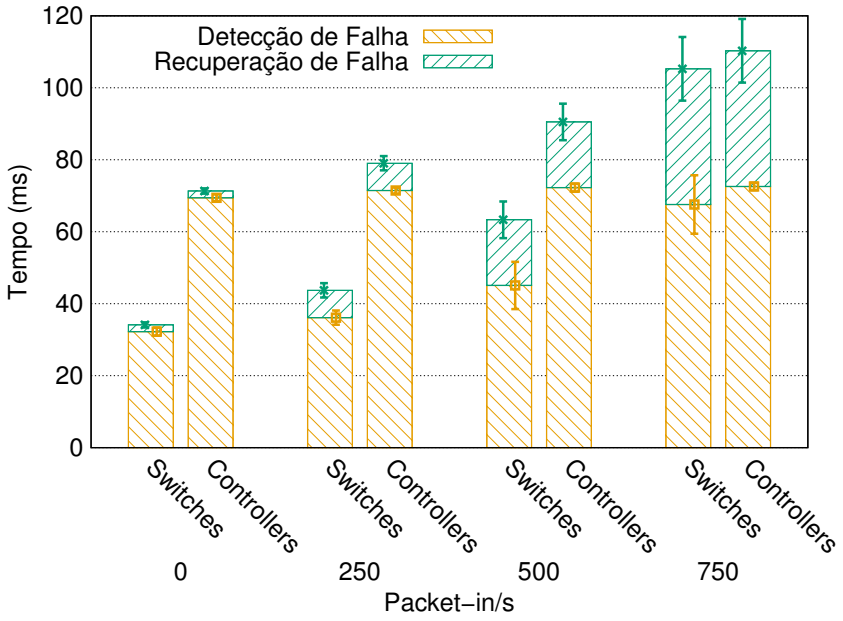


Figura 6.8: Latência detecção e recuperação falha - *switch versus* controlador.

Na recuperação de falha completa, em geral, a latência do tempo de detecção de falha é a componente dominante. Nos experimentos realizados os valores para este processo variaram entre 34,10 e 105,26 ms quando a detecção de falha foi executada pelo plano de dados. Os resultados com a detecção de falha executada pelo plano de controle variaram entre 71,30 e 110,27 ms.

Em relação à detecção de falhas nos controladores, a segunda proposta, executada pelo plano de dados, é mais eficiente, chegando a reduzir mais de 50% o tempo gasto para detectar uma falha nos controladores. Essa técnica tende a se manter mais eficiente enquanto o *switch* não é sobrecarregado, o que pode ser visto até a taxa de 500 *packet-in/s*. A partir de 750 *packet-in/s* os tempos das duas técnicas se equivalem. No entanto, com 1.000 *packet-in/s*, a detecção realizada utilizando os *switches* (via plano de dados) ultrapassou a técnica do plano de controle, ficando com latência aproximadamente 3,6 vezes maior. Esse comparativo pode ser visto na Figura 6.9.

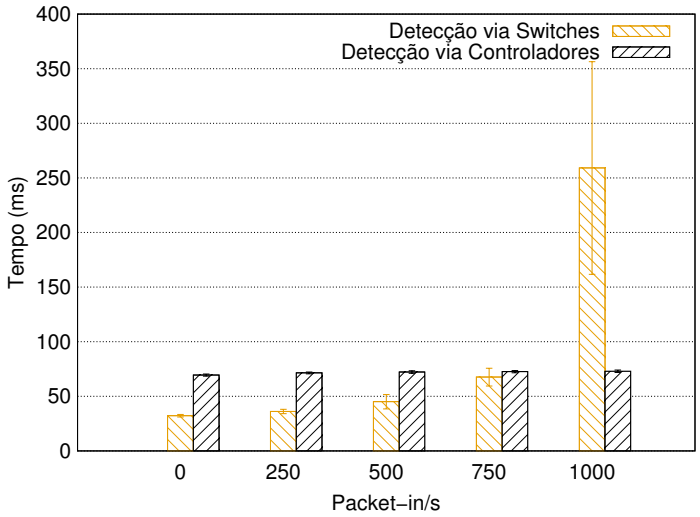


Figura 6.9: Detecção de falha - Switch x Controlador.

6.2.2 Avaliação Latência com Múltiplos Controladores

Em sistemas distribuídos, é importante destacar o comportamento do ambiente à medida em que nós são adicionados ao arranjo, e como o desempenho é impactado com a variação da carga. Devido ao número limitado de equipamentos disponíveis em nosso laboratório, utilizamos o ambiente Mininet para compor um cenário com mais controladores e *switches*. Nesta série de experimentos múltiplos controladores são utilizados, e então, mede-se a latência dos *packet-in/s* à medida em que o tráfego e o número de controladores/réplicas variam. Iniciamos os experimentos com dois controladores e chegamos a cinco, como número máximo.

A Tabela 6.1 apresenta os dados sobre a relação do número de controladores e a carga do sistema. O tráfego variou entre 250 e 1.000 *packet-in/s* por controlador, assim, o

throughput da rede é calculado como número de controladores multiplicado pela taxa de *packet-in/s*, e como esperado, um maior número réplicas implica em maior latência para processamento de *packet-in*.

A distribuição da tabela permite compararmos configurações equivalentes. Assim, podemos citar alguns resultados interessantes: no cenário com dois controladores e taxa de 500 *packet-in/s*, e no cenário com quatro controladores e taxa de 250 *packet-in/s*, apesar de apresentarem o mesmo *throughput*, o cenário com dois controladores apresenta uma latência menor. Quando há dois controladores e taxa de 750 *packet-in/s*, e três controladores com taxa de 500 *packet-in/s*, o experimento com dois controladores apresenta latência de 1,104 ms, e o com 03 (três) controladores, latência de 1,354 ms por pacote processado. Esses resultados indicam que à medida em que a quantidade de dados a serem replicados aumenta, por conta de replicação baseada em máquinas de estados, há um reflexo na latência de processamentos dos pacotes.

Tabela 6.1: Latência média em milisegundos de acordo com o número de controladores e a taxa de *packet-in/s* por controlador.

# de Controladores	**250 *packet-in*/s**	**500 *packet-in*/s**	**750 *packet-in*/s**	**1000 *packet-in*/s**
2	0,729 ±0,079	0,972 ±0,160	1,104 ±0,174	1,126 ±0,224
3	1,257 ±0,193	1,354 ±0,132	1,881 ±0,402	2,252 ±0,381
4	1,384 ±0,124	2,561 ±0,566	3,376 ±0,980	8,362 ±1,517
5	1,813 ±0,396	2,698 ±0,213	4,723 ±0,788	13,204 ±0,928

Outro achado do experimento é a observação de que as latências aumentam bastante quando há mais de três réplicas a partir da taxa de 500 *packet-in/s*. Por exemplo, no experimento com três controladores e taxa de 1.000 *packet-in/s* a latência é de 2,552 ms; com quatro controladores sobe para 8,362 *ms*. Neste exemplo, apesar do arranjo com quatro controladores processar 1.000 pacotes a mais que o arranjo com três, isto é, um *throughput* 33,3% maior, a latência foi aproximadamente 371% mais alta.

A maior diferença na relação Replicação x Desempenho, em configurações equivalentes, deu-se no arranjo com quatro e com dois controladores e *throughput* de 2.000 *packet-in/s*. Nesse arranjo, a latência no arranjo com dois controladores foi de 1,126 ms, e com quatro controladores foi de 2,561 ms, uma diferença de 1,435 ms.

6.2.3 Comentários finais

Algumas considerações podem ser feitas em relação aos resultados dos experimentos. De forma 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* ser maior que na recuperação de falha. 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 aumenta consideravelmente a latência.

No ambiente real, os *switches*, nos cenários a partir de 750 *packet-in/s* e 540 mpbs, operaram com grande carga de processamento, o que causou grande variação nos resultados. Isso foi 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 o 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, à medida que as taxas variaram, o aumento do consumo de CPU ocorreu de maneira gradual. Nestes experimentos realizados com as RouterBoards fica claro como é custoso encaminhar pacotes para o plano de dados. A partir dessa observação, chamamos atenção para o quão importante é conhecer o tráfego da rede em uma implementação OpenFlow, já que uma configuração equivocada, por exemplo, para o tempo de expiração das regras no *switch* pode gerar tráfego desnecessário para o plano de controle, consumindo a capacidade de processamento dos equipamentos.

Um outro importante levantamento diz respeito a latência de *packet-in*. Durante os experimentos que foram realizados antes e após as falhas dos controladores, as latências aumentaram de forma gradual à medida em que a taxa de pacotes gerados variava, iniciando em 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 8.200 μs. Os resultados apresentados neste experimento reforçam a importância do plano de controle ser escalável.

No processo de recuperação de falhas, ao incorporarmos o mecanismo de detecção de falhas, fica evidente a importância da escolha de uma estratégia eficiente. Ao compararmos os mecanismos de detecção de falhas realizados em ambos, *switches* e controladores, observamos que primeira técnica mostrou-se mais eficiente, sendo até 50% mais rápida que a segunda. Entretanto, destacamos que à medida que a taxa de ocupação da CPU dos *switches* aproxima-se do limite, o tempo de detecção da falha passa a ser maior, quando comparado com o mecanismo realizado pelos controladores.

Na discussão sobre a relação confiabilidade *versus* desempenho, deve-se levar em conta, mais uma vez, o levantamento sobre as características da rede a ser implementada e a sua sensibilidade a falhas. Nos experimentos realizados com múltiplos controladores, neste trabalho, em todos os cenários com diferentes números de controladores, mas com o mesmo *throughput*, a latência do arranjo com menor número de controladores foi mais baixa. Dessa forma, apesar de mais controladores significar maior confiabilidade, há perda

de desempenho à razão em que aumenta-se o número de réplicas.

Também foi observado que nos experimentos com dois e três controladores, à medida que a taxa de envio de pacotes para o plano de controle variava, as latências foram incrementadas de maneira gradual. Já quando havia quatro e cinco controladores na rede, à medida que a taxa de envio de *packet-in/s* foi incrementada, as latências variaram com mais intensidade.

Por fim, lembramos que o experimento com múltiplos controladores (cinco controladores no total) foi realizado em ambiente emulado, tendo como objetivo servir de referência, e que a partir das experiências obtidas neste trabalho, caso este arranjo seja implementado em ambiente real, a tendência é que haja um aumento para os valores dos resultados.

Capítulo 7

Conclusão e Trabalhos Futuros

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 e emulado, utilizando elementos de rede de prateleira (COTS) e o emulador de redes Mininet, 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 falhas, ficou caracterizado que a segunda, em geral, tem maior latência. No processo de detecção de falha, é evidente que a detecção, quando realizada no plano de dados, reduz sensivelmente a latência em relação à técnica quando utilizada no plano de controle. 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. Também observamos a relação confiabilidade x desempenho variando o número de controladores e a carga da rede, onde a partir de três controladores, a latência para processar *packet-in* aumenta substancialmente, o que pode indicar a região limite em que confiabilidade e desempenho se equivalem.

Para trabalhos futuros, pretendemos utilizar a capacidade de troca de *switches* entre controladores de forma a explorar balanceamento de carga no plano de controle, já que o protótipo possibilita a migração de *switches*. Na mesma linha, pretendemos habilitar os

controladores a trabalhar com uma visão parcial da rede, compatível com a visão global, de modo a reduzir a carga de trabalho em cada controlador. Sobre o novo modelo proposto para detecção de falha dos controladores pelo *switches*, o próximo passo é incorporar o mecanismo ao OvS. Com essa última alteração, temos expectativa de que o tempo para detecção de falha possa ser reduzido.

Referências Bibliográficas

[Botelho et al. 2014] 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 et al. 2013] 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 et al. 1993] Budhiraja, N., Marzullo, K., Schneider, F. B., and Toueg, S. (1993). The primary-backup approach. *Distributed systems*, 2:199–216.

[Chang et al. 2008] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4.

[Corradi et al. 2012] Corradi, A., Fanelli, M., and Foschini, L. (2012). Vm consolidation: A real case based on openstack cloud. *Future Generation Computer Systems*, (0):–.

[CPqD 2014] CPqD, F. (2014). Openflow 1.3 software switch. Website. https://github.com/CPqD/ofsoftswitch13.

[Dean and Ghemawat 2008] Dean, J. and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *CACM*, 51(1):107–113.

[DeCandia et al. 2007] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. (2007). Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM.

[Défago et al. 2004] 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 et al. 2013] 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.

[Fernandes 2013] Fernandes, E. L. (2013). Nox 1.3 oflib.

[Fonseca et al. 2013] 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.

[Forum 2013] Forum, T. M. E. (2013). Carrier ethernet management information model. Website. https://www.mef.net/Assets/Technical_Specifications/PDF/MEF_7.2.pdf.

[Foundation 2012] Foundation, O. N. (2012). Software-defined networking: The new norm for networks. *ONF White Paper*.

[Ghemawat et al. 2003] Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM.

[Gude et al. 2008] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). NOX: towards an operating system for networks. *Computer Communication Review*, 38(3):105–110.

[Handigol et al. 2012] Handigol, N., Heller, B., Jeyakumar, V., Lantz, B., and McKeown, N. (2012). Reproducible network experiments using container-based emulation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pages 253–264, New York, NY, USA. ACM.

[Handley et al. 2005] Handley, M., Kohler, E., Ghosh, A., Hodson, O., and Radoslavov, P. (2005). Designing extensible ip router software. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 189–202. USENIX Association.

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

[Hunt et al. 2010] 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] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., et al. (2010). Onix: A distributed control platform for large-scale production networks. *OSDI, Oct.*

[Kreutz et al. 2015] 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 1978] Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565.

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

[Lantz et al. 2010] Lantz, B., Heller, B., and McKeown, N. (2010). A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM.

[Liberato et al. 2014] 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.

[McKeown 2009] McKeown, N. (2009). Software-defined networking. *INFOCOM keynote talk, Apr*.

[McKeown et al. 2008] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G. M., Peterson, L. L., Rexford, J., Shenker, S., and Turner, J. S. (2008). Openflow: enabling innovation in campus networks. *Computer Communication Review*, 38(2):69–74.

[Mikrotik 2014] Mikrotik (2014). Mikrotik routeros. Website. http://www.mikrotik.com/pdf/what_is_routeros.pdf.

[ONF 2014] ONF, O. N. F. (2014). Openflow switch specification version 1.5.0 (wire protocol 0x06).

[OpenDaylight 2013] OpenDaylight (2013). Opendaylight sdn controller platform (oscp):overview. Website. https://wiki.opendaylight.org/view/OpenDaylight_SDN_Controller_Platform_(OSCP):Overview.

[OpenDayLight 2015] OpenDayLight (2015). Opendaylight "helium" – the rise of open sdn. "http://www.opendaylight.org/software".

[OpenWRT 2014] OpenWRT (2014). Openwrt.

[Panda et al. 2013] Panda, A., Scott, C., Ghodsi, A., Koponen, T., and Shenker, S. (2013). Cap for networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, HotSDN '13, pages 91–96, New York, NY, USA. ACM.

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

[Pfaff et al. 2009] Pfaff, B., Pettit, J., Amidon, K., Casado, M., Koponen, T., and Shenker, S. (2009). Extending networking into the virtualization layer. In [Subramanian et al. 2009].

[Rao 2015a] Rao, S. (2015a). Nox,the original openflow controller. Website. http://thenewstack.io/sdn-series-part-iii-nox-the-original-openflow-controller.

[Rao 2015b] Rao, S. (2015b). Ryu, a rich featured open source sdn controller supported by ntt labs. Website. http://goo.gl/WMbhSU.

[Rothenberg et al. 2010] 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.

[RYU 2014] RYU, P. T. (2014). Ryu sdn framework using openflow 1.3. Website. http://osrg.github.io/ryu-book/en/Ryubook.pdf.

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

[Sherwood et al. 2009] Sherwood, R., Gibb, G., Yap, K., Appenzeller, G., Casado, M., McKeown, N., and Parulkar, G. (2009). Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep.*

[Specification 2009] Specification, O. S. (2009). Version 1.0.0 (wire protocol 0x01).

[Specification 2011a] Specification, O. S. (2011a). Version 1.1.0 implemented.

[Specification 2011b] Specification, O. S. (2011b). Version 1.2.0 (wire protocol 0x04).

[Specification 2012] Specification, O. S. (2012). v1.3.0.

[Specification 2013] Specification, O. S. (2013). 1.4.0.

[Subramanian et al. 2009] Subramanian, L., Leland, W. E., and Mahajan, R., editors (2009). *Eight ACM Workshop on Hot Topics in Networks (HotNets-VIII), HOTNETS '09, New York City, NY, USA, October 22-23, 2009*. ACM SIGCOMM.

[Team 2013] Team, R. P. (2013). Ryu sdn framework. Website. http://osrg.github.io/ryu-book/en/Ryubook.pdf.

[Tourrilhes et al. 2014] Tourrilhes, J., Sharma, P., Banerjee, S., and Pettit, J. (2014). The evolution of sdn and openflow: A standards and perspective. *IEEE Digital Library*.

Publicações do Autor

1. SPALLA, E. ; MAFIOLETTI, D. ; LIBERATO, A. ; Rothenberg C. ; CAMARGOS, L. ; VILLACA, R. ; MARTINELLO, MAGNOS . Estrategias para Resiliencia em SDN : Uma Abordagem Centrada em Multi-Controladores Ativamente Replicados. In: XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2015, Vitoria-ES. Anais do SBRC, 2015.

2. SPALLA, E. ; MAFIOLETTI, D. ; LIBERATO, A. ; VILLACA, R. ; MARTINELLO, M. . Resiliência no Plano de Controle para Redes Definidas por Software. In: IV Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF), 2014, Florianópolis. Simpósio Brasileiro de Redes de Computadores - SBRC, 2014.

3. LIBERATO, A. ; MAFIOLETTI, D. ; SPALLA, E. ; VILLACA, R. ; MARTINELLO, M. . Avaliação de Desempenho de Plataformas para Validação de Redes Definidas por Software. In: Wperfomance, 2014, Brasília. WPerformance - XIII Workshop em Desempenho de Sistemas Computacionais e de Comunicação, 2014.

4. LIBERATO, A. ; SPALLA, E. ; MAFIOLETTI, D. ; VILLACA, R. ; MARTINELLO, M. . Balanceamento de Carga em SDN Provido por Comutadores Estocásticos de Prateleira. In: IV Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF), 2014, Florianópolis. Simpósio Brasileiro de Redes de Computadores - SBRC, 2014.

5. MARTINELLO, M. ; VILLACA, R. ; LIBERATO, A. ; MAFIOLETTI, D. ; SPALLA, E. ; CABELINO, R. . Plataformas Abertas para Infraestruturas Definidas por Software : Projeto, Implementação e Experimentos. In: Infraestruturas Definidas por Software (IDS) no WRNP, 2014, Florianópolis. Workshop da Rede Nacional de Ensino e Pesquisa (WRNP), 2014.