Ramon Marques Ramos

Roteamento Resiliente em Redes de Data Center utilizando OpenFlow: Uma abordagem centrada em caminhos embarcados na origem

Vitória

23 de agosto de 2013

Ramon Marques Ramos

Roteamento Resiliente em Redes de Data Center utilizando OpenFlow: Uma abordagem centrada em caminhos embarcados na origem

Dissertação apresentada para a obtenção do Grau de Mestre em Informática pela Universidade Federal do Espírito Santo.

Orientador: Prof. Dr. Magnos Martinello

Coorientador: Prof Dr. Christian Esteve Rothenberg

Universidade do Espírito Santo

Departamento de Informática

Programa de Pós-Graduação Informática

Vitória

23 de agosto de 2013

Dissertação de Mestrado sob o título "Roteamento Resiliente em Redes de Data Center utilizando OpenFlow: Uma abordagem centrada em caminhos embarcados na origem", defendida por Ramon Marques Ramos e aprovada em 19 de julho de 2013, em Vitória, Estado do Espírito Santo, pela banca examinadora constituída por:

Prof. Dr. Magnos Martinello
Orientador

Prof. Dr. Christian Esteve Rothenberg
Co-orientador

Prof. Dr. José Gonçalves Pereira Filho
Examinador

Prof. Dr. Moises Renato Nunes Ribeiro
Examinador

Dedicatoria

A todos aqueles que, de alguma forma, contribuíram para a conclusão deste trabalho.

Agradecimentos

Primeiramente ao orientador Magnos Martinello, pelos ensinamentos passados ao longo desses anos, pelas orientações e esforço para que eu pudesse superar meus limites.

Ao Professor Christian Esteve Rothemberg que contribuiu de forma primordial na evolução do trabalho.

Agradeço à minha namorada Bianca Honda Gerlin pelo apoio nos momentos mais difíceis.

Por fim, à minha família, que ao longo de toda minha vida deu todas as condições para que eu pudesse chegar até aqui. Sem eles nada disso teria sido possível.

Resumo

Redes de *Data Center* são um componente essencial da infraestrutura da Internet de hoje. Porém, abordagens de redes tradicionais não foram projetadas para atender as necessidades atuais de um ambiente virtualizado e não são flexíveis o suficiente para suportar as novas exigências. Recentemente, um conceito promissor de redes programáveis referido como Redes Definidas por Software (SDN - *Software Defined Networks*) está emergindo. SDN é uma proposta de arquitetura baseada em atributos como separação dos planos de dados e de controle, uma interface uniforme independente de fornecedor para o mecanismo de encaminhamento (ex: OpenFlow), um plano de controle logicamente centralizado e capacidade de virtualizar a rede física subjacente. Neste contexto, utilizando a visão global oferecida pelo controle centralizado, o roteamento na origem tem sido proposto como um meio para fornecer a escalabilidade, flexibilidade no encaminhamento e simplicidade no plano de dados. Este trabalho propõe uma abordagem de roteamento resiliente na origem utilizando OpenFlow para redes de *data center*, com base em caminhos codificados inseridos no cabeçalho dos pacotes. Propomos ainda duas abordagens na recuperação de falhas, um mecanismo de restauração de caminhos que envolve o controlador para trocar a rota dos fluxos e, posteriormente, a fim de diminuir o tempo de recuperação de falhas, um mecanismo de proteção, inserindo caminhos alternativos no cabeçalho dos pacotes. Avaliamos nossa proposta implementando um protótipo baseado em Open vSwitch e demonstramos sua eficácia em um cenário emulado no Mininet com topologias comuns de *data center*, *fat-tree*, BCube e DCell. A avaliação da proposta levou em conta o tempo de recuperação de falhas, requisitos de estado, desempenho do encaminhamento, e balanceamento de carga.

Palavras-chaves: *Redes*, *Redes definidas por software*, *Openflow*, *Recuperação de falhas.*

Abstract

Data Center Networks are an essential component of today's Internet architecture. However, traditional network approaches are simply not designed to meet the current needs of a virtualized environment and are not flexible enough to support the new demands. Recently, a promising concept of programmable networks called Software-Defined Networks (SDN) is emerging. SDN is an architectural proposal based on attributes such as separation of the data and control planes, an uniform interface for vendor-independent routing mechanism (e.g. OpenFlow), a logically centralized control plane and ability to virtualize the underlying physical network. In this context, using the global view offered by centralized controller, source routing has been proposed to provide scalability, flexibility and simplicity in the forwarding on the data plane. This paper proposes a resilient source routing approach to data center networks using OpenFlow, based on encoded paths inserted in the packet header. Furthermore, we propose two approaches for failure recovery, a mechanism of path restoration, involving the controller to change the route of flows, and subsequently, to decrease the recovery time, a path protection mechanism, inserting alternative paths in the packet headers. We evaluate our proposal by implementing a prototype based on the Open vSwitch and demonstrate its effectiveness in a scenario emulated in Mininet with fat-tree, BCube and DCell topologies. The evaluation takes into account the failure recovery time, performance analysis, state requirements, and load balancing.

Keywords: *Networks*, *Software defined networks*, *Openflow*, *failure recovery.*

Sumário

1 Introdução **17**

1.1 Contexto 17
1.2 Contribuição 19
1.3 Estrutura 20

2 Fundamentação Teórica **22**

2.1 Redes de *Data Center* 22
2.1.1 Comoditização das redes de *data center* 23
2.1.2 Arquiteturas de *data center* 24
2.1.3 Técnicas de encaminhamento 24
2.1.3.1 Encaminhamento Ethernet 25
2.1.3.2 Encaminhamento híbrido IP/Ethernet 26
2.1.3.3 VLANs 27
2.1.3.4 MPLS 27
2.1.4 Requisitos de redes de *data center* 28
2.2 Redes Definidas por Software 29
2.2.1 Limitações de tecnologias de rede atuais 29
2.2.2 Arquitetura das SDNs 30
2.2.3 OpenFlow 32
2.2.4 *Switch* OpenFlow 33
2.2.4.1 Tabela de fluxos 33
2.2.4.2 Ações do *switch* OpenFlow 35
2.2.4.3 Canal Seguro OpenFlow 35
2.2.4.4 Protocolo OpenFlow 36
2.2.5 Controlador 37
2.2.6 Implementação em *software* 38
2.2.7 Evolução do OpenFlow 39
2.3 Confiabilidade da rede 40
2.3.1 Recuperação de falhas de rede 40
2.3.1.1 Ciclo de recuperação 40
2.3.1.2 Ciclo de reversão 42
2.3.2 Mecanismos de recuperação 43
2.3.2.1 Caminhos de recuperação: pré-calculado x dinâmico . . . 43
2.3.2.2 Proteção x Restauração 43
2.3.2.3 Recuperação global x local 44

2.3.2.4 Controle de mecanismos de recuperação: Centralizados x Distribuídos . . . 44
2.4 Considerações do Capítulo . . . 45

3 Trabalhos Relacionados 46
3.1 Novas topologias . . . 46
3.1.1 Fat-trees . . . 46
3.1.2 B-Cube . . . 47
3.1.3 D-Cell . . . 48
3.2 Projetos de redes de *Data Center* . . . 49
3.2.1 VL2 . . . 49
3.2.1.1 Arquitetura . . . 49
3.2.1.2 Endereçamento e roteamento . . . 50
3.2.2 PortLand . . . 51
3.2.2.1 *Fabric Manager* . . . 51
3.2.3 Endereços PMAC (*Pseudo* MAC) . . . 52
3.3 Roteamento na origem . . . 52
3.3.1 *Switching with in-Packet Bloom Filters* (SiBF) . . . 53
3.3.1.1 Filtro de Bloom . . . 53
3.3.1.2 Encaminhamento . . . 54
3.3.1.3 *Rack Manager* . . . 55
3.3.2 *Slick Packets* . . . 56
3.4 Contribuições para o estado da arte . . . 56

4 Caminhos codificados em redes de *Data Center* 58
4.1 Visão Geral . . . 58
4.1.1 Princípios de projeto . . . 58
4.2 Descoberta da topologia e cálculo das rotas . . . 61
4.3 Encaminhamento com EP . . . 61
4.4 Implementação . . . 62
4.5 Resolução de requisições ARP . . . 64
4.6 Balanceamento de carga . . . 64
4.7 Roteamento tolerante a falhas . . . 64
4.8 Avaliação . . . 66
4.8.1 Tamanhos dos cabeçalhos . . . 66
4.8.2 Tempo de recuperação de falhas . . . 68
4.8.3 Desempenho de encaminhamento . . . 69
4.8.4 Balanceamento de carga . . . 70
4.8.5 Análise de requisitos de estado . . . 71

5 Roteamento na origem com caminhos alternativos 73
5.1 Visão Geral . . . 73
5.1.1 Princípios de projeto . . . 73
5.2 Seleção de caminhos . . . 74
5.3 Codificação . . . 75
5.4 Encaminhamento . . . 76
5.5 Implementação . . . 78
5.6 Avaliação . . . 79
5.6.1 Tamanho do cabeçalho codificado . . . 79
5.6.2 Tempo de recuperação de falhas . . . 81
5.6.3 Desempenho de encaminhamento . . . 82

6 Conclusão e Trabalhos Futuros 84
6.1 Contribuições . . . 84
6.2 Trabalhos Futuros . . . 86

Referências 87

Publicações do Autor 94

Lista de ilustrações

Figura 1 Interconexão de rede em supercomputadores. (TOP500, 2013) 24
Figura 2 Topologia genérica de redes de *data center*. (AL-FARES; LOUKISSAS; VAHDAT, 2008) 25
Figura 3 Arquitetura atual de roteadores. 31
Figura 4 Arquitetura das Redes Definidas por *Software*. 32
Figura 5 Arquitetura do *switch* OpenFlow. (CONSORTIUM, 2011) 33
Figura 6 Tabela de fluxo OpenFlow. (GUEDES et al., 2012) 34
Figura 7 Entradas na tabela de fluxo. 35
Figura 8 Ciclo de recuperação. (VASSEUR; PACKAVET; DEMEESTER, 2004) 41
Figura 9 Ciclo de reversão. (VASSEUR; PACKAVET; DEMEESTER, 2004) . . 42

Figura 10 *Fat-tree* utilizando *switches* de 4 portas. 47
Figura 11 BCube de nível 1 com n=4. (GUO et al., 2009) 48
Figura 12 DCell de nível 1 com n=4. (GUO et al., 2008) 49
Figura 13 Topologia Clos usada pelo VL2. (GREENBERG et al., 2009) 50
Figura 14 Encaminhamento de pacotes em uma rede VL2. (GREENBERG et al., 2009) 51
Figura 15 Mapeamento de MAC para PMAC. (MYSORE et al., 2009) 53
Figura 16 Arquitetura do SiBF. (ROTHENBERG et al., 2010) 54
Figura 17 Encaminhamento utilizando o SiBF. (ROTHENBERG et al., 2010) . . 55
Figura 18 Visão Geral do *Slick Packets*. (NGUYEN et al., 2011) 57

Figura 19 Visão geral do projeto. 60
Figura 20 Encaminhamento com EP. 62
Figura 21 Posição do EP no cabeçalho. 63
Figura 22 Recuperação de falhas. Após a falha, em vermelho, o caminho codificado no switch de origem será alterado pelo controlador. 66
Figura 23 Tamanho do cabeçalho EP para alguma topologias utilizadas em *data center*. 67
Figura 24 Tempo de recuperação com e sem caminhos pré calculados. 68
Figura 25 Comparação do tempo de recuperação em diversas topologias. 69
Figura 26 Desempenho do encaminhamento 70
Figura 27 Avaliação do Balanceamento de carga. CDF da utilização dos enlaces . 71

Figura 28 Visão geral do SlickFlow. 75

Figura 29 Codificação do EP com caminhos alternativos. 76
Figura 30 Exemplo do cabeçalho codificado 77
Figura 31 Tamanho do cabeçalho SlickFlow. 81
Figura 32 Tempo de recuperação. 81
Figura 33 Desempenho do encaminhamento. 82

Lista de tabelas

Tabela 1 Tamanho de cabeçalhos(em *bits*) para diferentes configurações. 67
Tabela 2 Configurações de topologias. 67
Tabela 3 Análise de Estado. 72

Tabela 4 Tamanho de cabeçalhos(em *bits*) para diferentes configurações. 80
Tabela 5 Configurações de topologias. 80

1 Introdução

1.1 Contexto

O surgimento de serviços populares tais como e-mail, serviços de busca e redes sociais, associado ao aumento da conectividade através de banda larga e redes ópticas, impulsionou o modelo de comunicação centrado no servidor. Cada vez mais o processamento e o armazenamento estão sendo movidos dos PCs (*Personal Computers*) para grandes provedores de serviços. Fotos, vídeos e aplicações que antes eram armazenadas e processadas nos PCs, agora migram para provedores sob o formato de serviços Web. Hoje, grandes empresas tais como Google, Yahoo!, Facebook e Microsoft permitem que aplicações sejam hospedadas e executadas remotamente em *data centers* (Amazon, 2013; CARR, 2006; DEAN; GHEMAWAT, 2008).

Assim, Redes de Centro de Dados (DCN - *Data Center Networks*) são um componente essencial na infraestrutura da Internet atual. O planejamento adequado do seu projeto de infraestrutura é crítico, exigindo requisitos de desempenho, resiliência, escalabilidade, entre outros (VERDI et al., 2010). *Data centers* estão crescendo em tamanho e importância à medida que muitas empresas estão migrando seus aplicativos e dados para a nuvem (CISCO, 2007). Para alcançar os benefícios do compartilhamento de recursos, centros de dados em nuvem devem escalar para tamanhos muito grandes com baixo aumento de custos. O crescimento contínuo dessas redes requer uma infraestrutura confiável, porque as interrupções nos serviços podem ter consequências significativas para as empresas.

DCNs de grande escala são normalmente compostos de vários racks de PCs *commodities*. Cada rack tem um *switch Top-of-Rack* (ToR), sendo os ToRs normalmente conectados a uma rede hierárquica interna composta de *switches* e roteadores de agregação do núcleo (CISCO, 2004). Dependendo de sua finalidade, os centros de dados variam em tamanho de centenas a centenas de milhares de máquinas (TAVAKOLI et al., 2009).

Para a construção de DCNs, pode-se utilizar *hardware* e protocolos de comunicação especializados, como InfiniBand (HAMADA; NAKASATO, 2005), que proporcionam escalabilidade e largura de banda. No entanto, a utilização de equipamentos especializados encarece muito o projeto, além de não ser nativamente compatível com aplicações TCP/IP. Por razões de compatibilidade e custos é cada vez mais comum que grandes redes de centros de dados empreguem *switches* Ethernet *commodities* de baixo custo para

interligar os *hosts*.

Apesar de sua importância, as arquiteturas de *data center* de hoje ainda estão longe de serem ideais. Os protocolos de roteamento, encaminhamento e gerenciamento que são executados atualmente foram projetados, de uma forma geral, para redes locais e se revelam insuficientes ao longo de uma série de dimensões para atender as demandas de computação em nuvem (KIM; CAESAR; REXFORD, 2008). Além do mais, a ausência de flexibilidade no controle do funcionamento interno dos equipamentos, assim como o alto custo da infraestrutura vigente, são barreiras para a evolução das arquiteturas e para a inovação decorrente da oferta de novos serviços e aplicações de rede.

A fim de se ter uma DCN ágil e não fragmentada, idealmente, a malha de conexão deveria comportar-se como um grande domínio Ethernet. Entretanto, a utilização do protocolo Ethernet voltado para LANs depende de inundações (*flooding*) em toda a rede para localizar *hosts* finais e faz uso de protocolos baseados em *broadcast*, como ARP, resultando em grande exigência de estado e sobrecarga de mensagens de controle, que crescem com o tamanho da rede. Outro ponto fraco da arquitetura Ethernet é a dependência do protocolo *Spanning Tree* (STP). Enquanto o STP funciona bem para pequenas redes, ele introduz ineficiências substanciais em redes maiores, que têm requisitos mais exigentes para baixa latência, alta disponibilidade e capacidade fim-a-fim (*bisection bandwidth*). Já o roteamento IP garante o uso eficiente e flexível de recursos de rede através de encaminhamento pelo caminho mais curto, mas leva a uma enorme sobrecarga de configuração e dificulta a atribuição arbitrária de máquinas virtuais (VMs) para qualquer servidor físico.

As abordagens tradicionais de redes simplesmente não foram projetadas para atender as necessidades atuais de um ambiente virtualizado e não são flexíveis o suficiente para suportar as novas exigências. Recentemente, um promissor conceito de redes referido como Redes Definidas por Software (SDN - *Software Defined Networks*) está emergindo. SDN é uma proposta de arquitetura baseada em um número de atributos-chave, incluindo separação dos planos de dados e de controle, uma interface uniforme independente de fornecedor para o mecanismo de encaminhamento (ex: OpenFlow (MCKEOWN et al., 2008)), um plano de controle logicamente centralizado e capacidade de virtualizar a rede física subjacente (GUDE et al., 2008).

Nesse modelo, considerando os controladores de SDN como o sistema operacional da rede, o *software* desenvolvido para criar novas funcionalidades pode ser visto como a aplicação que executa sobre um controlador que coordena a infraestrutura física. Valendo-se da visão unificada dos recursos de rede que o controlador oferece, pode-se implementar, por exemplo, soluções de roteamento especialmente desenhadas para uma topologia e com requisitos particulares.

Dessa forma, desenvolvedores podem definir o comportamento e novas funcionalidades da rede conforme os objetivos e requisitos específicos do provedor da infraestrutura

e serviços, sem dependerem dos demorados ciclos de atualização dos equipamentos de rede. Por isso, não é surpresa que provedores de infraestrutura para *cloud data centers*, tais como Google e Amazon, analisem com grande interesse a tecnologia OpenFlow (MILLER, 2011). A grande razão para este interesse reside nos níveis de controle e flexibilidade que eles precisam no aprovisionamento dos serviços na nuvem a um custo inferior, mas com desempenho equivalente dos *switches* comoditizados atualmente em operação.

1.2 Contribuição

Motivado por esse cenário de desafios e oportunidades, a contribuição do presente trabalho está no projeto, na implementação e na avaliação de uma proposta de roteamento resiliente em uma arquitetura de redes de *data center* controlada pelo protocolo OpenFlow. A idéia da proposta apoia-se em uma abordagem de roteamento centrada em caminhos embarcados nos cabeçalhos dos pacotes com base na codificação de rotas na origem.

Partindo da premissa que a rede do *data center* é administrada por uma entidade única, empregando os princípios de SDN, propomos um controlador logicamente centralizado que define regras nos *switches* de origem e destino. Embute-se informações do caminho no cabeçalho dos pacotes na origem, e retira-se no destino para entregar o pacote aos *hosts* finais. As informações que serão embutidas especificam um conjunto de nós no *data center* através dos quais os pacotes devem ser encaminhados.

A vantagem desta abordagem baseada em roteamento na origem é que a tarefa de encaminhamento de cada *switch* é simplificada, uma vez que requer apenas o conhecimento dos seus vizinhos, ao invés de uma entrada na tabela de encaminhamento para cada destino potencial. Além disso, como o primeiro comutador especifica a rota do pacote, os múltiplos caminhos disponíveis podem ser explorados.

No trabalho, a resiliência no roteamento foi abordada de duas formas. Primeiro, através de um mecanismo de restauração, onde um caminho alternativo é estabelecido pelo controlador quando uma notificação de falha é recebida dos *switches*. Como os caminhos são pré-computados, quando uma falha ocorrer em qualquer enlace da topologia, um outro caminho com nós disjuntos pode ser rapidamente selecionado, apenas embutindo um caminho diferente no cabeçalho dos pacotes, sem a necessidade de algoritmos distribuídos de convergência pela rede.

Posteriormente, propomos outra forma de recuperação de falhas, embutindo caminhos alternativos diretamente no cabeçalho do pacote. Se um enlace do caminho principal não estiver disponível, os *switches* podem enviar o pacote para um caminho alternativo. Este mecanismo de proteção permite que os comutadores redirecionem os pacotes diretamente no plano de dados, sendo uma alternativa mais rápida comparada à restauração de caminho utilizando o controlador.

A metodologia adotada no presente trabalho foi dividida em duas etapas. Primeiramente implementamos e avaliamos um conjunto de mecanismos de encaminhamento e roteamento para redes de *data center*, incluindo o mecanismo de recuperação de falhas baseado no controlador. Posteriormente, propomos e implementamos uma extensão da idéia, permitindo que os *switches* mudem a rota dos pacotes quando houver falhas na rede, com os caminhos alternativos sendo transportados no cabeçalho dos pacotes.

Neste trabalho utilizamos a versão 1.0 do OpenFlow. Para implementar o mecanismo de encaminhamento proposto, o software do *switch* OpenFlow foi modificado para fazer a manipulação dos caminhos nos cabeçalhos dos pacotes. O mecanismo de proteção com caminhos alternativos também requereu modificações adicionais nos commutadores para fazer a troca dos caminhos.

Um protótipo foi implementado como prova de conceito usando *switches* OpenFlow baseados em *software* dentro do Mininet, uma plataforma de emulação de Redes Definidas por *Software*. Os testes foram feitos sobre algumas das topologias mais utilizadas nas principais propostas recentes de DCN: *fat-tree*, DCell e BCube. Para a validação da implementação, experimentos foram conduzidos considerando métricas como o tempo de recuperação de falhas, balanceamento de carga e desempenho do encaminhamento.

1.3 Estrutura

Este trabalho está estruturado como segue:

- O Capítulo 2 apresenta os conceitos básicos, características e necessidades referentes à área de redes de *data center*. Em seguida, apresentamos os conceitos de Redes Definidas por *Software*, e sua realização através do OpenFlow. Por fim, mostramos os conceitos e características relativas aos mecanismos recuperação de falhas em redes de dados.

- O Capítulo 3 discute os trabalhos relacionados, analisando-os a fim de reforçar a justificativa e a relevância do trabalho que está sendo proposto. Além de apresentar características e métodos que serviram de fonte de inspiração para o presente trabalho.

- No Capítulo 4 apresentamos a proposta de roteamento e encaminhamento para DCNs baseada em caminhos codificados, e implementamos o mecanismo de recuperação de falhas que utiliza o controlador para trocar os caminhos dos pacotes. São descritos os principais blocos funcionais, a implementação e avaliação.

- No Capítulo 5 descrevemos e implementamos o mecanismo de recuperação de falhas com caminhos alternativos no cabeçalho dos pacotes, e por fim, é feita a avaliação do protótipo.

- Finalmente, o Capítulo 6 traz as conclusões e as perspectivas dos trabalhos futuros.

2 Fundamentação Teórica

O objetivo desse Capítulo é apresentar aspectos teóricos importantes para a compreensão deste trabalho. Serão abordados os principais conceitos das áreas de redes de comunicação em *data center* e recuperação de falhas em redes de dados, considerados necessários para a contextualização deste trabalho. Além disso, o capítulo também pretende dar a base necessária a respeito de Redes Definidas por *Software* (SDN) e OpenFlow, tecnologias nas quais o trabalho se baseia.

O capítulo está organizado da seguinte forma: a Seção 2.1 apresenta conceitos relacionados à área de redes de *data center*; a Seção 2.2 discute os conceitos de Redes Definidas por *Software* e sua realização utilizando OpenFlow, tecnologia sobre a qual a proposta foi implementada; a Seção 2.3 apresenta uma introdução aos mecanismos de recuperação em redes de comunicação; e, por fim, a Seção 2.4 apresenta algumas considerações sobre o capítulo.

2.1 Redes de *Data Center*

O modelo computacional dos centros de dados começou com os *mainframes* na década 1960. Posteriormente, após o surgimento dos microcomputadores, uma busca constante por altas capacidades de armazenamento se estabeleceu. Esta fase foi seguida pelos sistemas distribuídos baseados no modelo cliente/servidor e, subsequentemente, pelo crescimento explosivo da Internet e da Web. Mais recentemente, a evolução das técnicas de virtualização tem propiciado o desenvolvimento de aplicações que compartilham a mesma infraestrutura de *hardware*, alavancando o surgimento de soluções para serviços em nuvem.

Centros de dados em nuvem (*cloud data centers*), como o Amazon EC2 e Microsoft Azure estão se tornando cada vez mais populares, uma vez que oferecem recursos de computação a um custo muito baixo (MICROSOFT, 2013; NSF, 2013). Muitas pequenas e médias empresas estão se voltando para estes serviços, não só para a execução de grandes tarefas computacionais, mas também para seus sistemas e armazenagem de dados. Isso ajuda a eliminar a tarefa cara, e muitas vezes complexa, de construção e manutenção de infraestruturas próprias (MUDIGONDA et al., 2011).

Operadores de centros de dados em nuvem têm a capacidade de oferecer uma infraestrutura barata, pois podem compartilhar a infraestrutura física entre um grande número de inquilinos. O advento de técnicas de virtualização de CPU (por exemplo,

VMWare (WMWare Staff, 2013) e Xen (BARHAM et al., 2003)) torna possível a conversão dos servidores físicos dedicados, e muitas vezes extremamente subutilizados, em máquinas virtuais (VMs - *Virtual Machines*) que são executadas no *data center*.

Para usufruir plenamente dos benefícios do compartilhamento de recursos, essas redes devem escalar para grandes tamanhos, por exemplo, os chamados *mega data centers* têm na ordem de dezenas de milhares de servidores (VERDI et al., 2010). Quanto maior o número de inquilinos e máquinas virtuais, melhores serão as possibilidades de compartilhamento, o que por sua vez permite uma maior eficiência dos recursos, gerando redução dos custos (ARMBRUST et al., 2010).

A diversidade de aplicações que podem ser hospedadas usando o modelo em nuvem inclui desde aplicações comerciais, sistemas Web tradicionais até aplicações científicas para processamento paralelo e aplicações móveis. Estas diferentes aplicações requerem diferentes arquiteturas, e isto tem motivado a pesquisa e o desenvolvimento de soluções que atendam a esta demanda, no sentido de criar mecanismos escaláveis, com alto desempenho e custos mínimos. Isto inclui a pesquisa em infraestrutura voltada para a eficiência energética, cabeamento, resfriamento e, principalmente, infraestrutura de interconexão dos servidores (VERDI et al., 2010).

2.1.1 Comoditização das redes de *data center*

Existem duas abordagens de alto nível para a construção de redes de comunicação para grandes *clusters*. Uma abordagem utiliza protocolos de comunicação e *hardware* especializados para fornecer interconexão escalável de alto desempenho. Entre as tecnologias mais populares que se enquadram nessa categoria estão *InfiniBand* (HAMADA; NAKASATO, 2005) e *Myrinet* (BODEN et al., 1995). A principal desvantagem dessas abordagens é que elas não são nativamente compatíveis com os aplicativos usando TCP/IP (LOUKISSAS, 2008). Além disso, exigem interfaces não-padrão de rede para os *hosts* finais, o que eleva muito seu custo.

A segunda abordagem, que visa construir a rede a partir de equipamentos (servidores, sistemas de armazenamento e redes) menores e mais baratos, é conhecida como comoditização (*commoditization*), que faz uso principalmente de equipamentos baseados no padrão Ethernet. A motivação por trás dessa mudança é principalmente econômica, uma vez que a comoditização do centro de dados diminui significativamente o custo total de propriedade (TCO - *Total Cost of Ownership*) (Dell, 2010). Isso tornou possível a construção de *clusters* que consistem de centenas de milhares de nós, concebidos para satisfazer as necessidades crescentes de muitos tipos de aplicações, tanto em empresas como em laboratórios de pesquisa e universidades. Além disso, por geralmente utilizarem *switches* Ethernet, essas soluções são mais fáceis de instalar e gerenciar.

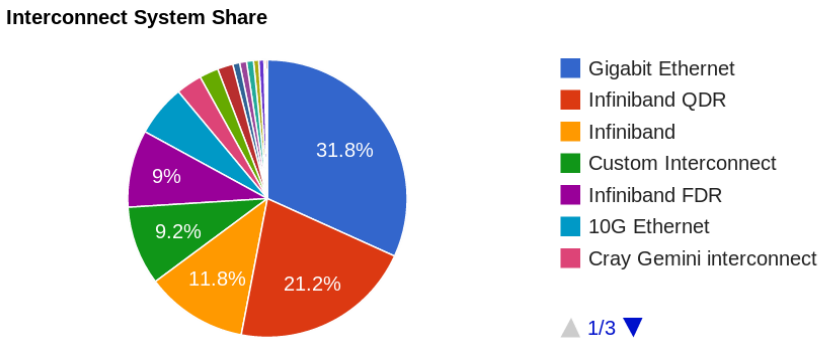


Figura 1: Interconexão de rede em supercomputadores. (TOP500, 2013)

Nos últimos anos, centros de dados *commodities* baseados no padrão Ethernet estão sendo amplamente utilizados e ganharam participação significativa em ambientes de computação de alto desempenho, como mostra a Figura 1.

2.1.2 Arquiteturas de *data center*

Tradicionalmente, a infraestrutura dos *data centers* é constituída por servidores físicos montados dentro de um rack e interligados através de um *switch* Ethernet de acesso, denominado ToR (*Top-of-Rack*). As redes apresentam duas ou três camadas, sendo que para *clusters* de maior escala geralmente se empregam três camadas, devido ao fato de que estas suportam mais *hosts*. Tipicamente uma rede de duas camadas suporta até 8 mil *hosts*, já uma rede de três camadas pode suportar mais de 25 mil *hosts* (AL-FARES; LOUKISSAS; VAHDAT, 2008).

Em uma rede de três níveis, os *switches* ToRs se interligam com um ou mais *switches* de agregação (AGGR), que agregam os *clusters* de servidores. Finalmente, estes se ligam a um terceiro nível de *switches*, categorizado como *switches* de núcleo (CORE), conforme mostra a topologia em camadas da Figura 2.

A abordagem de topologia em camadas é um fundamento básico dos *data centers* para prover escalabilidade, alto desempenho, flexibilidade, resiliência e facilidade de manutenção. Porém, a abordagem tradicional (topologias, protocolos de roteamento, separação por VLANs), que tem funcionado razoavelmente bem para os requisitos de redes corporativas, não é suficiente para atender os requisitos de DCNs de maior escala, como veremos a seguir.

2.1.3 Técnicas de encaminhamento

Operadores de rede atualmente enfrentam desafios significativos no gerenciamento e configuração dos grandes centros de dados (KIM; CAESAR; REXFORD, 2008). Estes problemas advém do fato de que os protocolos de rede tradicionais não conseguem atender

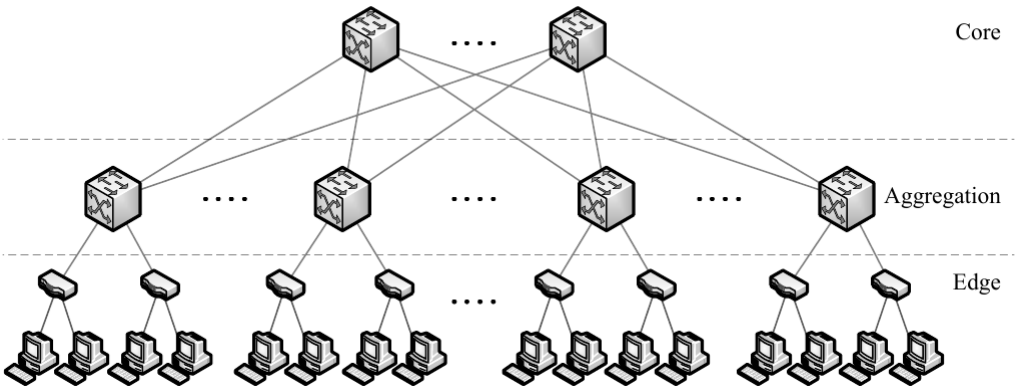


Figura 2: Topologia genérica de redes de *data center*. (AL-FARES; LOUKISSAS; VAHDAT, 2008)

por completo os requisitos das redes de grande escala. A dicotomia de alto nível é entre a criação de redes de camada 2 ou camada 3, cada um deles com vantagens e desvantagens associadas.

Para fornecer a base teórica necessária para o restante do trabalho, esta seção explica, primeiro, o motivo das redes Ethernet não serem uma alternativa viável para grandes escalas. Em seguida, descrevemos um modelo híbrido IP/Ethernet e VLANs, duas abordagens amplamente utilizadas que melhoram a escalabilidade do Ethernet convencional, mas introduzem suas próprias limitações. Por fim, discutiremos a utilização de MPLS (*Multi Protocol Label Switching*) em DCNs.

2.1.3.1 Encaminhamento Ethernet

O Ethernet se destaca como uma das tecnologias de rede mais utilizadas atualmente. Este protocolo tem características atraentes, especialmente para administradores de redes, devido sua simplicidade e facilidade de configuração (KIM; REXFORD, 2007).

Neste protocolo, os *hosts* finais já têm endereços globalmente únicos (endereços MAC de 48 bits), sem necessidade de nenhuma configuração. Além disso, os *switches* Ethernet aprendem automaticamente a localização dos *hosts*, reduzindo ainda mais a necessidade de configuração da rede. Ainda, os endereços planos permanentes simplificam o suporte à mobilidade, à resolução de problemas de rede, e à aplicação de políticas de controle de acesso. Logo, a arquitetura Ethernet "pura"seria extremamente atraente, não fossem suas sérias limitações de escalabilidade.

O Ethernet depende de transmissão *broadcast* para suportar serviços essenciais como o ARP (*Address Resolution Protocol*) e o DHCP (*Dynamic Host Configuration Protocol*). Isso consome recursos em excesso, e também apresenta vulnerabilidades de segurança (MYERS; NG; ZHANG, 2004).

A utilização do Ethernet geralmente exige um mecanismo de proteção de circuito, como STP (*Spanning Tree Protocol*). O protocolo *Spanning Tree* automaticamente evita *loops*, impedindo que pacotes *broadcast* circularem continuamente e travem a rede. Contudo, o STP leva a caminhos sub-utilizados e consequentemente distribuição irregular de cargas.

Além disso, o protocolo se baseia em *flooding* (inundação) para entregar os quadros Ethernet a destinos desconhecidos. A sobrecarga de controle necessária para divulgar informações de cada máquina através de *flooding* pode ser muito grande, desperdiçando largura de banda e recursos de processamento (KIM; CAESAR; REXFORD, 2008). Ademais, as tabelas de encaminhamento nos *switches* podem crescer muito, pois o plano de endereçamento aumenta o tamanho das tabelas proporcionalmente ao número total de *hosts* na rede.

2.1.3.2 Encaminhamento híbrido IP/Ethernet

Uma forma de aumentar a escalabilidade do Ethernet é a construção de redes com várias LANs interligadas através de roteamento IP. Nessas redes híbridas, cada LAN contém no máximo algumas centenas de *hosts* que formam uma sub-rede IP. A cada sub-rede é atribuído um prefixo IP que a representa, e para cada *host* é atribuído um endereço IP com o prefixo da sub-rede. Ao contrário de um endereço MAC, que funciona como um identificador do *host*, um endereço IP denota sua localização na rede. Tal atribuição vai permitir tabelas de encaminhamento relativamente pequenas em todos os *switches* do *data center*.

Protocolos de roteamento intra-domínio, como OSPF (MOY, 1998) podem ser empregados para encontrar os caminhos de menor custos entre os *hosts*. Falhas na rede em topologias de grande escala são comuns. O OSPF pode detectar essas falhas e, em seguida, transmitir as informações para todos os *switches* para assim, evitarem *links* indisponíveis. Infelizmente, essa arquitetura também impõe suas limitações.

O maior problema da arquitetura híbrida é a sua enorme sobrecarga de configuração. A atribuição de prefixos para as sub-redes, e a configuração das sub-redes nos roteadores normalmente são processos manuais, pois a atribuição deve seguir a hierarquia de endereçamento, e deve ser planejada de forma a reduzir o espaço de endereços desperdiçados, considerando também o uso futuro de endereços para minimizar uma reconfiguração posterior. Além do mais, o processo de adicionar um novo elemento de rede, geralmente, requer a configuração manual por parte do administrador de rede. O envolvimento de administradores humanos aumenta o tempo de reação a falhas e aumenta o potencial de erros de configuração.

Roteadores IP não funcionarão corretamente até que os administradores configurem suas sub-redes. Da mesma forma, um *host* final não pode acessar a rede até que

seja configurado com um endereço IP correspondente à sub-rede onde a máquina está atualmente localizada. A utilização de DHCP automatiza a configuração dos *hosts*, mas introduz uma sobrecarga considerável de gerenciamento. A falta de sincronia entre um servidor DHCP e um identificador de sub-rede de um roteador pode levar a *hosts* inacessíveis e erros difíceis de diagnosticar.

2.1.3.3 VLANs

Um meio termo entre uma rede de camada 2 e camada 3 consiste em empregar VLANs para permitir uma camada lógica de nível 2 atravessar múltiplos domínios de *broadcast*. Ao contrário de uma LAN física, uma VLAN pode ser logicamente definida, independentemente da localização dos *hosts* na rede. Ao dividir uma grande rede em várias VLANs de dimensões adequadas, os administradores podem reduzir a sobrecarga de transmissão de *broadcast* em cada uma delas, e também garantir o isolamento entre os diferentes grupos. Comparado com o IP, VLANs simplificam a mobilidade, pois os *hosts* mantém seus endereços IP enquanto circulam em uma mesma VLAN.

Embora viável para redes de menor escala, VLANs também sofrem de uma série de inconvenientes. Elas devem ser configuradas em cada *switch*, e exigem que os recursos de largura de banda sejam explicitamente atribuídos a cada VLAN em cada *switch* participante, limitando a flexibilidade para alterar dinamicamente os padrões de comunicação.

Embora VLANs reduzam a sobrecarga de transmissão *broadcast* sobre os *hosts*, *switches* configurados com várias VLANs devem manter as entradas em suas tabelas de encaminhamento e processar tráfego *broadcast* para cada *host* ativo em cada VLAN visível.

Ainda, as *spanning trees* utilizadas em cada VLAN impedem que certos links sejam usados. Embora a configuração de árvores geradoras disjuntas para cada VLAN (SHARMA et al., 2004) possam melhorar o balanceamento de carga e aumentar o *throughput*, uso eficaz de árvores requer que estas sejam movidas para que o equilíbrio seja mantido, o que deve ser feito manualmente, à medida que o tráfego muda (KIM; REXFORD, 2007).

2.1.3.4 MPLS

O MPLS é uma tecnologia baseada no encaminhamento de dados através de rótulos que sinalizam o caminho do pacote, evitando pesquisas em tabelas de encaminhamento. Os rótulos do MPLS possuem 20 bits, podendo suportar milhões de conexões. Assim o MPLS poderia prover tanto escalabilidade quanto encaminhamento eficiente. Porém, os argumentos contrários são principalmente em relação aos custos. Equipamentes de rede com suporte ao MPLS geralmente são caros, pois sua implementação é complexa (KOMPELLA, 2012), além de o suporte ser limitado. Além disso o custo operacional também

é maior pois o protocoloe é difícil de configurar, gerenciar e solucionar problemas de depuração.

2.1.4 Requisitos de redes de *data center*

Como podemos ver, a abordagem tradicional (topologia, protocolos de roteamento, organização hierárquica baseado nas camadas L2/L3, separação por VLANs), que tem funcionado razoavelmente bem para os requisitos de redes tradicionais, não é suficiente para atender os requisitos de grandes *data centers*.

Assim, é desejável que as novas arquitetura de centro de dados atendam a requisitos como (VERDI et al., 2010):

Agilidade: Agilidade é a capacidade de realocar dinamicamente servidores entre os serviços oferecidos pela nuvem com o propósito de otimizar a alocação da carga de trabalho (GREENBERG et al., 2009). As práticas atuais de engenharia do *data center* e, em especial, a organização hierárquica L2/L3 da arquitetura de rede, causam uma série de problemas que dificultam a agilidade do sistema. Sem agilidade, cada serviço deve pré-alocar a quantidade de servidores necessários para atender a demanda em momentos de picos de utilização de um determinado serviço;

Escalabilidade: é preciso que o roteamento e endereçamento sejam escaláveis, tabelas de encaminhamento (L2 Ethernet) tenham tamanho gerenciável e tempos de convergência sejam aceitáveis. O objetivo final é poder escalar a uma ordem de centenas de milhares de servidores, sendo assim, dois aspectos precisam ser considerados: o tamanho das tabelas de encaminhamento e o tráfego *broadcast*, dois grandes problemas para redes de grande escala;

Tolerância a falhas: o projeto da infraestrutura deve suportar falhas de servidores e *switches*. Devido ao grande número de equipamentos, as chances de falhas são maiores e precisam ser contornadas para que o sistema se mantenha operacional;

Múltiplos caminhos: O tráfego de rede deve ser capaz de usar todos os caminhos entre as partes comunicantes e evitar o congestionamento dos enlaces. Isto por sua vez significa remover o STP, que desperdiça largura de banda preciosa bloqueando muitos *links* para evitar *loops* de encaminhamento.

Configuração automática: O tamanho astronômico de *data centers* virtualizados torna o processo de configuração e administração manual da rede uma tarefa muito difícil e propensa a erros. Para evitar interrupções no serviço e fornecer agilidade, a DCN deve ser capaz de oferecer mecanismos *plug-and-play* .

Custo: melhorar os esforços de configuração através de mecanismos automatizados ajudam a reduzir o custo de manutenção dos sistemas. Usar *hardware* comoditizado

reduz o custo de capital. Isto envolve também novos métodos de estruturação e cabeamento, procurando modularidade da infraestrutura do *data center*, que contribuam para uma redução dos gastos. Além disso, melhorar a eficiência energética é fazer com que, idealmente, o consumo de energia seja proporcional à carga de trabalho, um problema bastante desafiador.

Cada um destes requisitos pode ser resolvido separadamente com as tecnologias de rede atuais. Porém, satisfazer todos eles ao mesmo tempo é um grande desafios, principalmente porque atender a um determinado requisito implica, muitas vezes, em não conseguir atender a outro.

Conclui-se, baseado nos comentários anteriores, que as técnicas convencionais não oferecem o desempenho nem a agilidade e flexibilidade na gerência do endereçamento IP/Ethernet. Vale a pena destacar que estas limitações não são resolvidas simplesmente com um aumento da capacidade de transmissão dos enlaces.

2.2 Redes Definidas por Software

2.2.1 Limitações de tecnologias de rede atuais

Comparando as redes de computadores de 20 anos atrás com as atuais fica claro que um longo caminho foi percorrido e muito se evoluiu em termos de velocidade, confiabilidade, segurança e abrangência. As tecnologias de camada física evoluíram proporcionando comunicação de alta capacidade e os dispositivos de rede melhoraram no poder computacional, proporcionando o surgimento de uma grande quantidade de aplicações de rede inovadoras, como *MapReduce* (DEAN; GHEMAWAT, 2008), sistemas de armazenamento de arquivos distribuídos (GHEMAWAT; GOBIOFF; LEUNG, 2003; CHANG et al., 2006), serviços de Internet (DECANDIA et al., 2007) e aplicações científicas que utilizam processamento paralelo (CHEVERESAN et al., 2007). Porém, a rede em sua estrutura não passou por muitas mudanças.

Na infraestrutura existente, tarefas que compõem a funcionalidade geral da rede, como roteamento ou decisões de acesso são delegadas a equipamentos fechados de vários fabricantes diferentes, cada um executando seus próprios *firmwares*.

Esta base bem estabelecida de equipamentos com *softwares* proprietários que compõem toda a infraestrutura de rede não abre muito espaço para novas idéias, como pesquisa de novos protocolos de roteamento, que precisam ser testados em larga escala em redes reais (SHERWOOD et al., 2010). A penetração da rede em vários setores cruciais do nosso cotidiano desencoraja qualquer tentativa de experimentação com o tráfego de produção, sendo esta uma das principais razões pelas quais a infraestrutura de rede se manteve estática e inflexível e nenhum grande avanço foi alcançado nessa direção.

Além disso, o modelo convencional de redes impõe limitações aos seus administradores. Qualquer mudança de configuração avançada do equipamento, na especialização da lógica de controle e tratamento dos pacotes ou ainda, inserção de novas funcionalidades estão sujeitas a ciclos de desenvolvimento e de testes restritos ao fabricante do equipamento, resultando em um processo demorado e custoso (HAMILTON, 2011).

A ausência de flexibilidade no controle do funcionamento interno dos equipamentos, assim como o alto custo da infraestrutura vigente, são barreiras para a evolução das arquiteturas e para a inovação decorrente da oferta de novos serviços e aplicações de rede.

A partir das diversas propostas que visavam a centralização da inteligência de controle da rede, tais como redes ativas (TENNENHOUSE; WETHERALL, 1996), o modelo 4D (GREENBERG et al., 2005) e o Ethane (CASADO et al., 2007), surgiu a arquitetura de Redes Definidas por *Software* (SDN - *Software Defined Networks*). Em SDNs, os planos de dados e de controle são separados, a inteligência de rede e as informações de estado são logicamente centralizadas, e a infraestrutura da rede subjacente é abstraída a partir das aplicações. Como resultado, empresas e operadoras ganham poder de programação, automação e controle, permitindo-lhes construir redes altamente escaláveis e flexíveis que se adaptam facilmente às necessidades de negócios em constante mudança.

SDN é uma arquitetura de rede dinâmica e flexível que protege os investimentos existentes, pois promove o uso de interfaces abertas de *software* para permitir o desenvolvimento de aplicativos de terceiros, que podem melhorar ou fornecer novos serviços de rede, além de permitir a orquestração de recursos entre múltiplos dispositivos, como *switches*, roteadores, *firewalls* e balanceadores de carga.

Esta proposta tecnológica é especialmente interessante para as redes de *data center*, de forma que os próprios provedores podem definir a funcionalidade da rede, com um baixo custo do equipamento e, especialmente customizado para suas necessidades de escala e controle.

Com SDN, as redes estáticas de hoje podem evoluir para uma plataforma extensível de entrega de serviços capaz de responder rapidamente às mudanças de necessidade do negócio, do usuário final, e do mercado.

Através de um esforço integrado da comunidade acadêmica e da indústria (Open Networking Foundation, 2012), as SDNs estão transformando a arquitetura de rede de dados, deixando-as mais flexíveis, fáceis de operar e gerenciar, e mais capazes de responder às novas exigências de aplicações e condições de rede.

2.2.2 Arquitetura das SDNs

Para entender as SDNs, primeiro vamos analisar o modelo tradicional de redes (Figura 3). Uma análise da arquitetura atual dos roteadores permite observar que se

trata de um modelo formado basicamente por duas camadas bem distintas: o *software* de controle e o *hardware* dedicado ao encaminhamento de pacotes (Juniper Networks, 2010). O primeiro, encarregado de tomar as decisões de roteamento, transfere essas decisões para o plano de encaminhamento através de uma API proprietária. A única interação da gerência com o dispositivo ocorre através de interfaces de configuração (Web, SNMP, CLI, por exemplo), limitando o uso dos dispositivos às funcionalidades programadas pelo fabricante (ROTHENBERG et al., 2011).

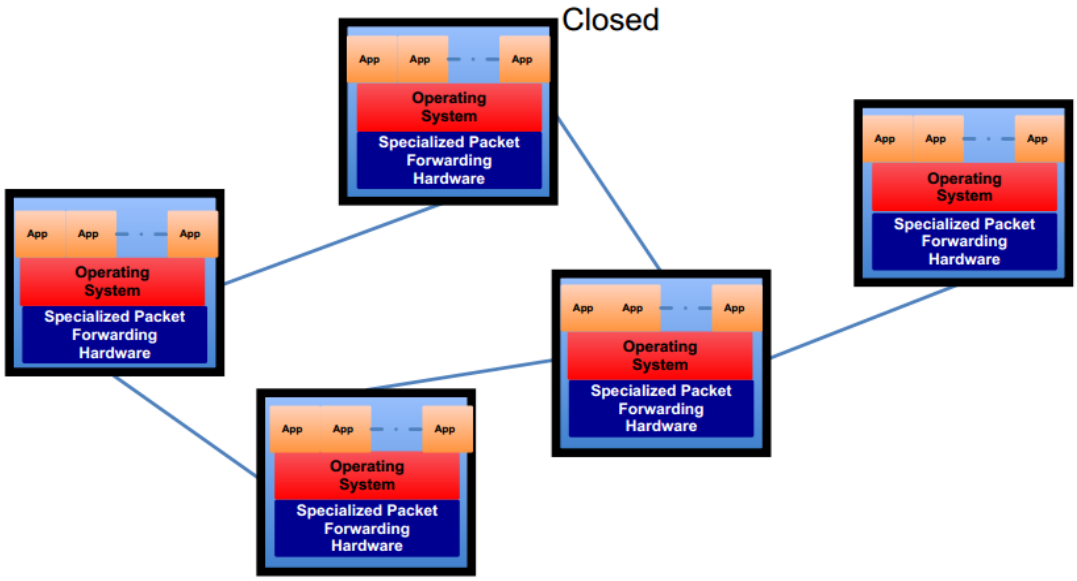


Figura 3: Arquitetura atual de roteadores.

É coerente pensar que se a arquitetura é, atualmente, composta por duas camadas autocontidas, estas não precisam estar fechadas em um mesmo equipamento. Para isso, basta que exista uma forma padrão de se programar o dispositivo de rede remotamente, permitindo que a camada de controle possa ser movida para um servidor dedicado e com alta capacidade de processamento. Desse modo, mantém-se o alto desempenho no encaminhamento de pacotes em *hardware* aliado à flexibilidade de se inserir, remover e especializar aplicações em *software* por meio de um protocolo aberto para programação da lógica do equipamento.

A Figura 4 mostra uma visão lógica da arquitetura de SDN. A inteligência de rede é (logicamente) centralizada em controladores baseados em *software*, que mantêm uma visão global da rede. Como resultado, a rede pode ser vista pelo plano de controle como um único *switch* lógico. Isso possibilita maior controle sobre toda a rede, o que simplifica seu projeto e operação. SDN também simplifica os próprios dispositivos, uma vez que já não precisam de compreender e processar milhares de padrões de protocolo, mas simplesmente aceitar instruções dos controladores SDN.

Dessa forma, os operadores e administradores de rede podem configurar através

de *software*, essa abstração simplificada da rede ao invés de terem de fazê-lo manualmente em arquivos de configuração espalhados entre os dispositivos. Além disso, aproveitando a inteligência centralizada do controlador SDN, é possível alterar o comportamento da rede em tempo real e implantar novas aplicações e serviços em muito menos tempo. Centralizando o estado da rede na camada de controle, a arquitetura SDN flexibiliza a configuração, o gerenciamento e otimiza os recursos de rede através de *softwares* dinâmicos e automatizados. Os próprios operadores da rede podem escrever esses programas, sem precisar esperar que novos recursos sejam incorporados nos softwares proprietários dos fabricantes de dispositivos.

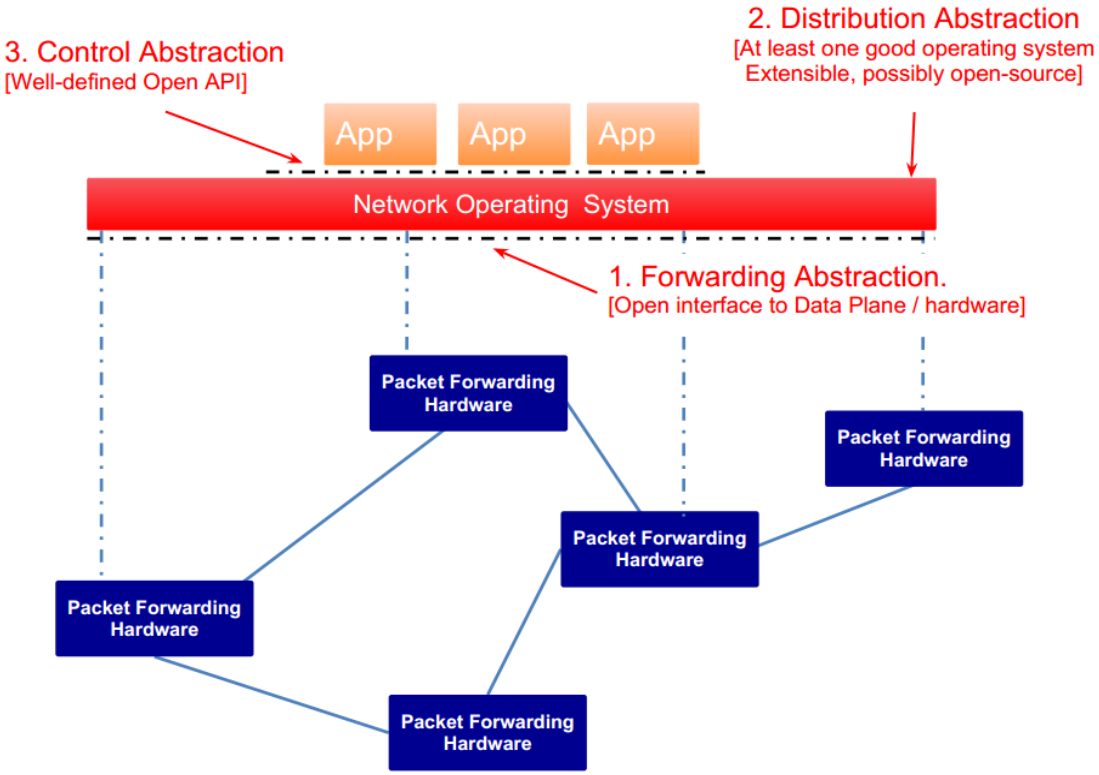


Figura 4: Arquitetura das Redes Definidas por *Software*.

2.2.3 OpenFlow

Proposto por um projeto iniciado na Universidade de Stanford, OpenFlow (MCKEOWN et al., 2008) é o primeiro padrão de interface de comunicação entre as camadas de controle e de encaminhamento para arquiteturas SDN, permitindo o acesso e manipulação do plano de encaminhamento de dispositivos de rede (Open Networking Foundation, 2012). Este padrão foi projetado com o objetivo permitir que se utilize equipamentos comerciais para construção de redes com a arquitetura SDN.

OpenFlow usa o conceito de fluxos para identificar o tráfego com base em regras (ações), que definem a forma como os pacotes devem fluir através dos dispositivos de

rede. As regras são instaladas no *hardware* de encaminhamento pelo controlador externo. Como o OpenFlow permite que a rede seja programada a nível de fluxos, uma arquitetura baseada nessa tecnologia proporciona um controle extremamente granular.

Uma rede OpenFlow é formada, basicamente, por *switches* OpenFlow, que contém uma ou mais tabelas de fluxo, e um ou mais controladores, que são os responsáveis por adicionar e remover as regras de manipulação de fluxo. Veremos agora esses elemento mais detalhadamente.

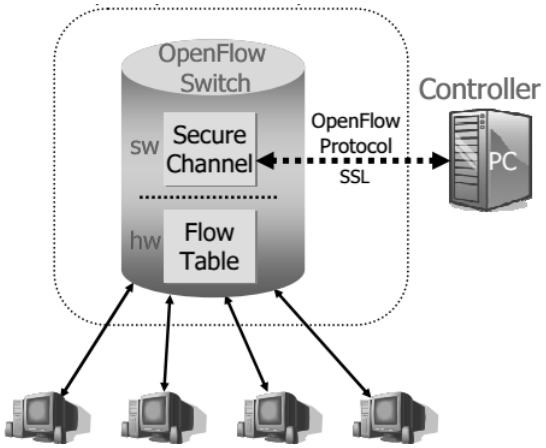


Figura 5: Arquitetura do *switch* OpenFlow. (CONSORTIUM, 2011)

2.2.4 *Switch* OpenFlow

A idéia básica do OpenFlow é explorar o fato de que os *switches* e roteadores contém tabelas de fluxo, normalmente construídas a partir de TCAM's, memórias que realizam pesquisas em alta velocidade. Embora as tabelas de fluxo de cada fornecedor sejam diferentes, foi identificado um conjunto de funções em comum que são executadas nesses dispositivos. O OpenFlow explora este conjunto de funções para fornecer um protocolo aberto para a programação das tabelas de fluxo em diferentes *switches* e roteadores.

Um *switch* OpenFlow consiste em pelo menos três partes (ver Figura 5): (1) Tabela de Fluxo (*Flow-Table*), com ações associadas a cada entrada de fluxo, para informar ao *switch* como processar os pacotes, (2) Um Canal Seguro (Secure Channel) que liga o *switch* ao controlador remoto, permitindo que os comandos e pacotes sejam transferidos entre eles usando o (3) o Protocolo OpenFlow, que fornece um padrão aberto para um controlador para se comunicar com um *switch*.

2.2.4.1 Tabela de fluxos

Um grande vantagem do OpenFlow é a flexibilidade que ele oferece para se programar o tratamento de cada fluxo, definindo como ele deve (ou não) ser encaminhado pela rede. O padrão determina como um fluxo pode ser definido e as ações que podem ser

realizadas sobre os pacotes pertencentes a ele. A união de uma definição de fluxo e um conjunto de ações formam uma entrada da tabela de fluxos

As tabelas de fluxos dos *switches* OpenFlow consistem de três campos, mostrados na Figura 6: (i) a combinação dos campos do cabeçalho do pacote que definem o fluxo; (ii) a ação que define como o fluxo será processado; e (iii) contadores usados para fins de estatística, que têm a finalidade de manter o controle do número de pacotes e bytes para cada fluxo e o tempo decorrido desde o início do fluxo.

Ao manter uma tabela de fluxo, o *switch* é capaz de tomar decisões de encaminhamento para os pacotes recebidos com uma pesquisa simples nas entradas da tabela, executando uma verificação de correspondência em campos específicos dos pacotes recebidos. Para cada pacote recebido, o *switch* busca em sua tabela de fluxos uma entrada correspondente, se possuir, o pacote é encaminhado com base na ação associada a esta entrada.

Nesse sentido, uma das contribuição mais importante do paradigma OpenFlow é a generalização do plano de dados. Qualquer modelo de encaminhamento de dados baseado na tomada de decisão fundamentada em algum valor, ou combinação de valores dos campos de cabeçalho dos pacotes pode ser suportado.

Deve-se enfatizar que a abstração da tabela de fluxos ainda está sujeita a refinamentos, com o objetivo de oferecer uma melhor exposição dos recursos do hardware.

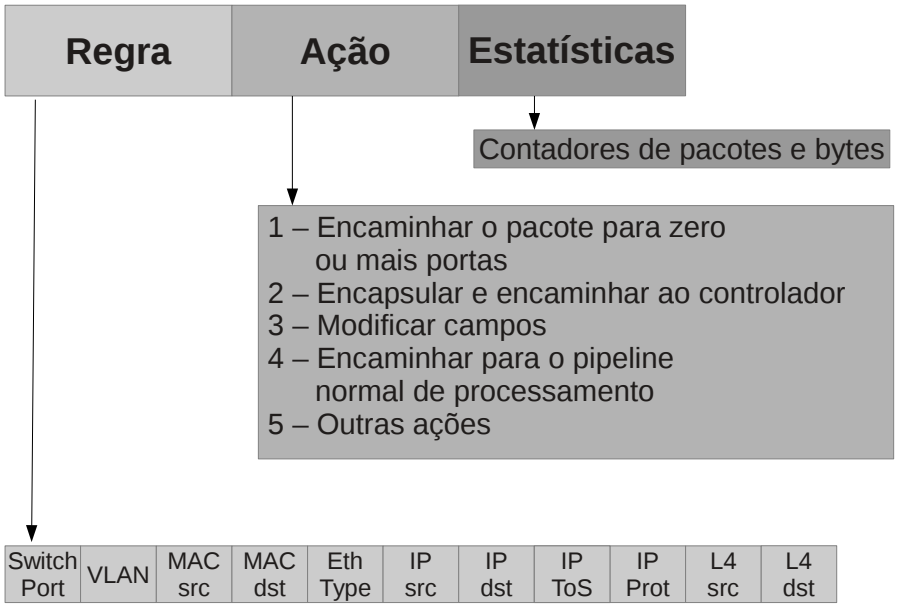


Figura 6: Tabela de fluxo OpenFlow. (GUEDES et al., 2012)

2.2.4.2 Ações do *switch* OpenFlow

O *switch* OpenFlow pode executar três diferentes tipos de ações associadas às entradas de fluxo da tabela e uma quarta se for um dispositivo baseado em *hardware*. As ações são as seguintes:

1. Encaminhar os pacotes para uma determinada porta (ou portas).
2. Encapsular e transmitir os pacotes para um controlador. O pacote é entregue ao canal seguro (*Secure Channel*), onde é encapsulado e enviado para um controlador. Tipicamente é o que acontece com o primeiro pacote de um novo fluxo, assim, o controlador pode decidir se uma entrada deve ser adicionada à tabela de fluxo do *switch*.
3. Descartar os pacotes do fluxo. Pode ser usado para segurança, para reduzir ataques de negação de serviço, ou para reduzir algum tipo de tráfego na rede.
4. Para os dispositivos baseados em hardware: Enviar os pacotes para o canal normal de processamento do *switch*.

Esse pequeno conjunto de ações cria diversas possibilidades, pois permite que muitas das funcionalidades que são implementadas separadamente possam ser agrupadas em um único controlador OpenFlow. Alguns exemplos das possibilidades são apresentadas na Figura 7. A primeira linha da tabela de fluxos representa o uso do *switch* OpenFlow para realizar encaminhamento de pacotes na camada de enlace, com base no endereço Ethernet de destino (MAC_{dst}). A segunda e terceira linhas implementam, respectivamente, um *firewall* para impedir conexões à um serviço SSH (porta 22) e atribuir uma tag VLAN para o fluxo destinado a um serviço Telnet (porta 23).

Switch Port	VLAN	MAC src	MAC dst	Eth Type	IP src	IP dst	IP ToS	IP proto	L4 src	L4 dst	Actions
*	*	*	00:1F:22:...	*	*	*	*	*	*	*	Port 4
*	*	*	*	*	*	*	*	TCP	*	22	Drop
*	*	*	00:1F:22:...	*	*	*	*	UDP	*	23	Set VLAN=1, Port 1

Figura 7: Entradas na tabela de fluxo.

2.2.4.3 Canal Seguro OpenFlow

O canal seguro é a interface que conecta cada *switch* OpenFlow ao controlador. Através desta interface, o controlador configura, gerencia e recebe eventos do *switch*. Para

que a rede não sofra ataques de elementos mal intencionados, o canal seguro garante confiabilidade na troca de informações entre o *switch* e o controlador. A interface de acesso recomendada é o protocolo SSL (Secure Socket Layer).

2.2.4.4 Protocolo OpenFlow

O protocolo OpenFlow é a linguagem usada pelos dispositivos OpenFlow para a comunicação entre eles. Este protocolo suporta três tipos de mensagens: **controlador-para-*switch***, **assíncronas** e as mensagens **simétricas**. Vamos descrever brevemente esses três tipos de mensagens. Para um estudo mais aprofundado, a especificação OpenFlow (MCKEOWN et al., 2008) fornece uma excelente fonte de informações.

As mensagens **controlador-para-*switch*** são iniciados pelo controlador e nem sempre exigem uma resposta. Através dessas mensagens o controlador configura o *switch*, gerencia as tabelas de fluxo e adquire informações sobre o estado dessas tabelas ou as capacidades suportadas pelo *switches* a qualquer momento.

Entre as mensagens mais importantes deste tipo estão:

- ***Flow-Mod***: são enviados pelo controlador para gerenciar o estado dos *switches*. Sua finalidade é a de adicionar, remover e modificar fluxos nas tabelas de encaminhamento e definir as propriedades das portas.
- ***Stats Request***: recuperam informações dos contadores de um *switch*.
- ***Features Request***: permitem ao controlador consultar parâmetros de configuração do *switch*, para saber, por exemplo, quais portas estão disponíveis.

As mensagens **assíncronas** são enviados do *switch* para o controlador e denotam uma mudança de estado do dispositivo ou da rede. Esta mudança também é chamada de evento. Um dos eventos mais significativos é o **packet-in** (chegada de pacote) que ocorre sempre que um pacote que não tem uma entrada de fluxo correspondente chega ao *switch*. Quando isto acontece, uma mensagem **packet-in** é enviada ao controlador contendo o pacote ou uma fração dele, a fim de que o controlador o examine e determine qual o tipo de fluxo deve ser estabelecido.

Listamos abaixo as principais mensagens assíncronas:

- ***Packet-in***: enviam ao controlador os pacotes que não fizeram *match* com nenhuma regra no *switch*.
- ***Port Status***: informam ao controlador a mudança no *status* de alguma porta do *switch*, ou seja, quando uma porta ativa se torna inativa, ou vice-versa.

- ***Flow Removed***: informam ao controlador que um fluxo foi removido devido seu tempo de vida ou de inatividade terem se esgotado.

- ***Error***: informam sobre erros ocorridos no *switch*.

Finalmente, a terceira categoria de mensagens OpenFlow são as mensagens **simétricas**, que são enviadas sem solicitação em qualquer direção. Estes podem ser utilizados para auxiliar ou diagnosticar problemas na comunicação do controlador com o *switch*.

- ***Hello***: são mensagens trocadas entre o controlador e o *switch* no momento da inicialização.

- ***Echo***: mensagens de eco podem ser enviadas tanto pelo *switch* quanto pelo controlador, e devem retornar uma resposta de eco. Eles podem ser usados para medir a latência ou a largura de banda de um enlace, bem como verificar se este está ativo.

- ***Experimenter***: mensagens *experimenter* fornecem uma maneira padrão de incluir mensagens adicionais no OpenFlow. Esta é uma espaço de teste para recursos destinados à futuras revisões.

2.2.5 Controlador

O controlador é o elemento que concentra as tarefas de manipulação direta dos dispositivos de rede e oferece uma abstração de nível mais alto para o desenvolvedor. Em uma analogia direta das funcionalidades de um sistema operacional, esse elemento age como um sistema operacional para a rede: provendo o controle direto dos dispositivos, oferecendo uma interface mais eficiente para os desenvolvedores, isolando os detalhes de cada componente.

O controlador pode, dessa forma, concentrar a comunicação com todos os elementos programáveis que compõem a rede e oferecer uma visão unificada do estado da rede. Sobre essa visão, comandos podem ser executados e outros programas, implementando novas funcionalidades, podem ser desenvolvidos (CASADO et al., 2010).

É importante observar que essa noção de visão única e centralizada da rede é uma visão lógica. Não necessariamente exige que o controlador opere como um elemento concentrador, fisicamente localizado em um ponto único do sistema. A abstração de visão global pode ser implementada de forma distribuída, como já foi proposto em trabalhos como (KOPONEN et al., 2010) e (TOOTOONCHIAN; GANJALI, 2010).

Atualmente, existem várias implementações de controladores disponíveis.

NOX (GUDE et al., 2008) é o controlador original OpenFlow e tem como principal função o desenvolvimento de controladores eficientes em C++. Devido a seu pioneirismo, grande parte dos projetos de pesquisa na área são baseados no NOX. Atualmente ele suporta a versão 1.0 do OpenFlow, mas existe uma modificação feita pelo CPqD que suporta até a versão 1.3 (CPQD, 2013).

POX é um *fork* do NOX desenvolvido com a premissa de ser completamente escrito em Python, resultando em uma interface mais elegante e simples. Os desenvolvedores de POX acreditam este controlador é adequado para substituir NOX nos casos em que Python é utilizado, enquanto NOX ainda permanece como um ambiente adequado para implementações que tenham demandas mais elevadas em termos de desempenho.

Onix (KOPONEN et al., 2010) é um controlador desenvolvido em parceria pela Nicira, Google e NEC, com o objetivo de permitir o controle de redes de grandes dimensões de forma confiável. Para garantir a escalabilidade e confiabilidade do sistema, Onix provê abstrações para particionar e distribuir o estado da rede em múltiplos controladores distribuídos, endereçando as questões de escalabilidade e tolerância a falhas, que surgem quando um controlador centralizado é utilizado.

Apesar de não se tratar de um controlador propriamente dito, vale citar o **Frenetic** (FOSTER et al., 2011), uma **linguagem de programação de redes** que tem o objetivo de facilitar a programação de controladores no contexto de redes OpenFlow. A idéia do Frenetic é prover abstrações de alto nível, dando aos programadores controle direto sobre a rede, evitando preocupações com detalhes de implementação de controladores em particular.

Em relação a emuladores e simuladores, é importante mencionar o **Mininet** (LANTZ; HELLER; MCKEOWN, 2010). Esta ferramenta de emulação de Redes Definidas por *Software* permite a rápida prototipação de redes utilizando apenas um computador, através de primitivas de virtualização do sistema operacional, como processos e *namespaces* de rede. Com essas primitivas, ele permite rapidamente criar, interagir e customizar protótipos de redes utilizando *switches* OpenFlow.

2.2.6 Implementação em *software*

A implementação de referência do modelo OpenFlow é um comutador em *software*, executando no espaço de usuário em uma máquina Linux. Esse modelo tem sido utilizado como base em diversos experimentos, mas carece de um melhor desempenho.

Desenvolvido para suprir essa lacuna, o *Open vSwitch* (OvS) é um *switch* virtual que segue a arquitetura OpenFlow, também é implementado em *software*, mas com o plano de dados dentro do kernel Linux, enquanto o plano de controle é acessado a partir do espaço de usuário (PFAFF et al., 2009).

Há também propostas para novas implementações que alteram a divisão de tarefas entre controlador e os switches, como no caso da arquitetura DevoFlow (CURTIS et al., 2011). O argumento dos autores é que no padrão OpenFlow tradicional, a necessidade de que todos os fluxos sejam acessíveis para o controlador SDN impõe demandas sobre o *hardware* e limitações de desempenho que podem não ser aceitáveis em alguns casos, em particular para fluxos de alto desempenho. Dessa forma, DevoFlow reduz o número de casos em que o controlador precisa ser acionado, aumentando a eficiência da rede.

Como é possível que dispositivos de rede suportem tanto o OpenFlow como o encaminhamento tradicional, foi muito fácil para as empresas e operadoras introduzirem progressivamente tecnologias SDN baseadas em OpenFlow. A tecnologia está sendo amplamente adotada por fornecedores de infraestrutura, que normalmente têm aplicado através de um *firmware* simples ou atualização de *software*.

2.2.7 Evolução do OpenFlow

A versão 1.0 do OpenFlow, lançada no fim de 2009, incluiu versões atualizadas da especificação do *switch* OpenFlow, a implementação de referência e ferramentas relacionadas. Nessa versão o controlador dispõe da abstração 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. Essa versão suporta ainda máscaras de bits nos campos do protocolo IP.

Na versão 1.1 do protocolo OpenFlow foi considerado o suporte a múltiplas tabelas de fluxos concatenadas, mediante um novo conjunto de instruções, bem como novas ações (copy/decrement TTL, push/pop tag, QoS) e campos de cabeçalho (VLAN priority, MPLS label/traffic class, SCTPport) para uma definição ainda mais completa dos fluxos e do conjunto de ações associadas. Foi considerado ainda o uso de máscaras de bits nos campos Ethernet.

Na versão 1.2 do protocolo, lançada no fim de 2011 se destaca a adição do suporte ao IPv6. E em abril de 2012, na versão 1.3, foi adicionado o suporte aos cabeçalhos de extensão do IPv6.

As versões 1.X do OpenFlow ainda possuem algumas limitações em termos do uso do padrão em circuitos ópticos e uma definição de fluxos que englobe protocolos que não fazem parte do modelo TCP/IP. No entanto, a versão 2.0 está sendo formulada e um dos objetivos e eliminar algumas dessas limitações.

Alguns switches comerciais já suportam o padrão OpenFlow, como e o caso do HP Procurve 5400zl, do NEC IP880 e do Pronto 3240 e 3290. Espera-se que à medida que a especificação OpenFlow evolua, e mais fabricantes de equipamentos de rede incorporem os padrões as suas soluções comerciais.

2.3 Confiabilidade da rede

As redes de comunicação estão sujeitas a uma grande variedade de falhas não intencionais causadas por desastres naturais (terremotos, incêndios e inundações), desgaste, sobrecarga, erros de *software*, erros humanos, e assim por diante, bem como falhas intencionais causados por ações de manutenção ou sabotagem . Essas falhas afetam as instalações de rede, como a infraestrutura de transmissão ou comutação, cujo fracasso, por sua vez, interrompe serviços de comunicação para empresas e usuários residenciais.

Para os clientes empresariais, a interrupção da comunicação pode suspender operações críticas, podendo causar uma perda significativa de receitas para o cliente. Na verdade, as garantias de disponibilidade (e compensações, caso não sejam cumpridas) formam agora um componente importante do nível de serviço (SLAs - *Service Level Agreements*) entre fornecedor e cliente (ITU-T, 2008). Por isso, é essencial fornecer um elevado nível de disponibilidade, isso é, a permanência das funções de rede necessárias para que os serviços de comunicação continuem funcionando (VASSEUR; PACKAVET; DEMEESTER, 2004).

2.3.1 Recuperação de falhas de rede

Muitos serviços exigem alta disponibilidade da rede, no entanto uma variedade muito grande de tipos de falha pode ocorrer em uma rede de dados Assim, deve-se evitar a ocorrência de falhas tanto quanto possível, e quando ocorrerem é preciso recuperar-se rapidamente. Assim, redes de comunicações modernas usam mecanismos de recuperação de falhas. Logo que que uma falha na rede é detectada, esses mecanismos desviam automaticamente o fluxo afetado pela falha para um outro caminho. Estes esquemas podem aumentar consideravelmente a disponibilidade dos serviços transportados através da rede.

O princípio básico de um sistema de recuperação é simples. Em condições normais (isto é, sem falhas), o tráfego é transportado ao longo do caminho primário. Se for detectada uma falha ao longo desse caminho, o esquema de recuperação é ativado. Uma parte do percurso (ou todo ele, dependendo da técnica de recuperação), será contornado por um caminho alternativo. Na maioria dos casos, é usado um caminho de recuperação disjunto do caminho principal, para assegurar que uma falha não afete ambos os caminhos (SHARMA; HELLSTRAND, 2003).

Embora existam uma grande variedade de esquemas de recuperação, todos eles apresentam uma sucessão de fases bastante semelhante, isto é, o ciclo de recuperação.

2.3.1.1 Ciclo de recuperação

As fases deste ciclo são apresentados na Figura 8. Se uma falha na rede ocorre, pode demorar algum tempo antes de um nó adjacente à falha detectá-la. Esse tempo pode

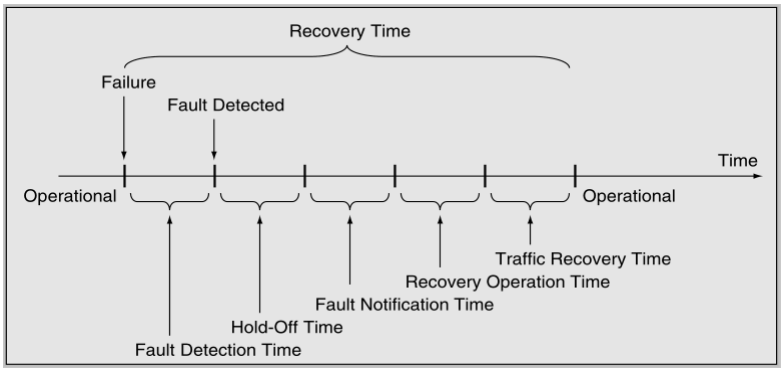


Figura 8: Ciclo de recuperação. (VASSEUR; PACKAVET; DEMEESTER, 2004)

depender, por exemplo, da frequência de sinais enviados, da velocidade de detecção de falha em uma camada inferior de rede e da notificação para camadas superiores, do tempo que um nó leva para coletar toda as informações de vários sinais e assim por diante.

Uma vez que a falha é detectada, o nó que a detectou pode (ou não) esperar algum tempo antes de começar a enviar mensagens de notificação para os outros nós da rede. Este tempo de espera poderia permitir que um esquema de recuperação de uma camada inferior reparasse a falha. Por exemplo, em uma rede IP sobre uma rede de transporte óptico, um corte do cabo podem ser rapidamente reparados por um mecanismo de recuperação óptico, assim, a comunicação IP se torna operacional novamente logo após o momento da falha. Se a falha persistir após o tempo de espera, mensagens de notificação são enviadas em toda a rede para informar aos outros nós que estarão envolvidos na ação de recuperação.

O tempo de espera pode ser de natureza estática, ou seja, um valor fixo que será usado em todos os casos, ou dinâmica. Neste último caso, o temporizador é uma função do número de falhas dentro de um determinado período. Quanto mais falhas forem detectadas, maior será o tempo de espera. Esta técnica é denominada de *amortecimento* e ajuda a estabilizar a rede no caso de um recurso instável (isto é, um recurso que alterne rapidamente entre o estado de funcionamento e estado de falha).

O tempo entre a primeira e a última ação de recuperação é chamado o tempo de operação de recuperação. Este intervalo de tempo pode incluir a troca de mensagens entre os diferentes nós envolvidos na ação de recuperação para coordenar a operação. Após a última ação de recuperação o tráfego começa a usar o caminho de recuperação. No entanto, ele ainda pode levar algum tempo antes que o tráfego esteja completamente recuperado. Este tempo de recuperação do tráfego pode depender do atraso de propagação ao longo do caminho de recuperação, a localização da falha, e o esquema de recuperação usado.

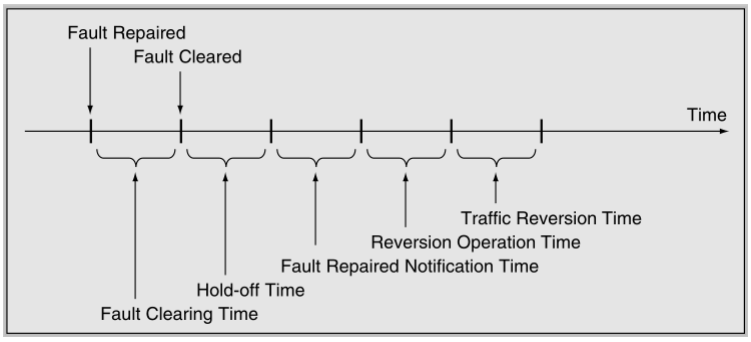


Figura 9: Ciclo de reversão. (VASSEUR; PACKAVET; DEMEESTER, 2004)

2.3.1.2 Ciclo de reversão

Após o ciclo de recuperação, a rede está mais uma vez completamente operacional. No entanto, as novas rotas do tráfego ao longo dos caminhos de recuperação podem não ser as ideais (por exemplo, o caminho de recuperação é mais longo ou mais congestionado que o caminho original). Portanto, um protocolo dinâmico de reencaminhamento pode ser iniciado para otimizar o uso dos recursos na nova situação. Outra possibilidade é a de esperar a reparação do defeito que tenha ocorrido e redirecionar o tráfego do caminho de recuperação de volta para o percurso original, uma vez que a falha esteja completamente reparada. Esta operação também segue uma sucessão de fases, o ciclo de reversão.

As diferentes fases deste ciclo são apresentados na Figura 9. O ciclo de reversão tem uma forte semelhança com o ciclo de recuperação, descrito anteriormente. Uma vez que a falha esteja reparada, pode demorar algum tempo (por exemplo, dependendo de protocolos de camada inferior) antes de este reparo ser detectado. Depois disso, o protocolo pode decidir esperar durante um certo tempo antes de iniciar a notificação de reparação do defeito. Este tempo pode ser necessário para assegurar que o caminho é estável, pois no caso de uma falha intermitente, uma reação rápida de processo de reversão pode conduzir a condições de rede instáveis. Tal como no caso anterior, note que o temporizador pode ser estático ou dinâmico (amortecimento).

Depois disso, de um modo semelhante como no ciclo de recuperação, a reparação da falha é notificada pela rede e a operação de reversão real é iniciada, o tráfego é comutado de novo a partir do caminho de recuperação para o percurso principal. Finalmente, pode levar algum tempo antes que o tráfego comece a fluir normalmente no caminho original (tempo de reversão de tráfego).

Em contraste com o ciclo de recuperação, que é tipicamente a reação a um acontecimento inesperado, a falha do ciclo de reversão pode ser planejada com antecedência. Em uma reversão, não há necessidade de uma operação precipitada, um mecanismo bem controlado com o mínimo de interrupção é normalmente preferido.

2.3.2 Mecanismos de recuperação

Existe uma grande variedade de mecanismos de recuperação, cada um com seus pontos fortes e fracos. A tecnologia de rede específica normalmente impõe algumas restrições sobre a viabilidade de implementação de um sistema de recuperação. No que se segue, são elucidados as escolhas essenciais na concepção de um mecanismo de recuperação, enumerando os prós e contras de cada opção (VASSEUR; PACKAVET; DEMEESTER, 2004).

2.3.2.1 Caminhos de recuperação: pré-calculado x dinâmico

Esta escolha se refere ao momento em que o caminho de recuperação é escolhido. Na opção de caminhos pré calculados, para todas as situações de falha, o caminho do fluxo de recuperação é calculado antecipadamente (isto é, antes de qualquer falha ocorrer). Na opção dinâmica caminhos de recuperação não são planejados, o seu caminho é calculado uma vez que a falha é detectada. Se ocorrer uma falha, o mecanismo de recuperação começa a procurar de forma dinâmica por caminhos alternativos possíveis em toda a rede.

A vantagem da opção de caminhos pré-calculados é que estes permitem uma rápida recuperação da falha, enquanto que um mecanismo de recuperação dinâmico pode demorar mais tempo para identificar caminhos de recuperação adequados. Por outro lado, a opção por caminhos pré-calculados carece de flexibilidade para cenários não planejados. Um mecanismo dinâmico é capaz de procurar caminhos de recuperação mesmo para cenários não contabilizados, embora não haja nenhuma garantia de que ele vai encontrar esse caminho de recuperação.

2.3.2.2 Proteção x Restauração

Uma distinção muito importante a ser feita ao considerar mecanismos de recuperação é entre proteção e restauração. Ambas as opções exigem sinalização, mas a sutil diferença está no momento das ações de sinalização. No caso da proteção, os caminhos de recuperação são pré-calculados e totalmente sinalizados antes de ocorrer uma falha. Assim, quando a falha ocorre, nenhuma sinalização adicional é necessária para estabelecer o caminho de proteção. No caso da restauração, os caminhos de recuperação podem ser pré-calculados ou dinâmicos, mas quando ocorre uma falha, será necessária sinalização adicional para estabelecer o caminho de restauração.

Uma grande vantagem da proteção em comparação com a restauração é tipicamente o tempo de recuperação rápida. A sinalização adicional, após uma ocorrência de falha no caso de restauração pode consumir um tempo precioso. Por outro lado, técnicas de restauração podem ser mais flexíveis em relação aos cenários de falha que podem ser recuperados.

2.3.2.3 Recuperação global x local

Para contornar os pontos de falha, esquemas de recuperação mudam a rota do tráfego afetado. Na recuperação local, apenas os elementos de rede afetados são contornados, alterando o mínimo possível a rota original do fluxo. Se um único enlace falhar, um caminho de recuperação se estabelece entre os nós adjacentes à falha. No outro extremo temos a recuperação global, onde o caminho completo entre a origem e o destino é trocado pelo caminho de recuperação.

Claro que essas duas abordagens representam apenas os dois extremos de um grande número de possibilidades intermediárias, onde a extensão da recuperação é maior que na abordagem local, porém mais curto na global. Por exemplo, em redes G-MPLS essas opções intermédias são chamadas de recuperação do segmento (MANNIE; PAPADIMITRIOU, 2006).

A recuperação local é, geralmente, mais rápida que a global, sendo essa uma vantagem importante em aplicações sensíveis à latência. Por outro lado, na recuperação local a rota resultante após a recuperação pode ser mais longa do que o caminho original. Além disso, se dois enlaces sucessivos no caminho principal falharem, a recuperação global ainda resolverá o problema, já a recuperação local, não.

2.3.2.4 Controle de mecanismos de recuperação: Centralizados x Distribuídos

Mecanismos de recuperação centralizados dependem de um controlador central para determinar quais ações de recuperação devem ser tomadas, a partir de uma visão global do estado da rede. Este controlador determina onde e quando uma falha ocorreu, reúne informações de estado de toda a rede e comanda a reconfiguração de todos os elementos envolvidos no processo de recuperação. Sistemas de gerenciamento baseados em Redes Definidas por *Software* geralmente empregam mecanismos centralizados.

Já **Mecanismos de recuperação distribuídos** operam sem a intervenção de um sistema de controle central. Neste caso, os elementos de rede possuem sistemas de controle inteligentes, que iniciam de forma autônoma as ações de recuperação. Assim, o controle é distribuída ao longo dos elementos de rede envolvidos no processo de recuperação. Em contraste com os mecanismos centralizados, estes sistemas de controle distribuído não têm uma visão global, mas apenas uma visualização local do estado da rede. Eles podem ter de trocar mensagens entre si para fornecer informações suficientes e coordenar as ações de recuperação. Assim, múltiplos elementos de rede trabalham em paralelo para alocar o tráfego interrompido em uma rota alternativa. Um exemplo típico de um sistema de distribuição é o plano de controle em redes IP e MPLS (SHARMA; HELLSTRAND, 2003).

2.4 Considerações do Capítulo

O modelo convencional de rede IP/Ethernet não atende os requisitos de custo, escala e controle necessários aos provedores de computação em grandes DCNs. É por este motivo e pelas características especiais das redes de *data center* que novos projetos e propostas têm emergido para atender os objetivos específicos, que são criticamente diferentes das redes locais e metropolitanas.

Com o surgimento de *switches* programáveis baseados na tecnologia OpenFlow, emerge um promissor conceito de Redes Definidas por *Software*. Esta proposta tecnológica é especialmente interessante para as redes de *data center*, de forma que os próprios provedores possam definir a funcionalidade da rede, com um baixo custo do equipamento e, especialmente customizado para as necessidades de escala e controle da rede.

Este capítulo apresentou conceitos básicos relacionados à área de redes de centros de dados, redes definidas por *software* e recuperação de falhas em redes de dados. Estes conceitos são necessários para o entendimento deste trabalho, visto que a proposta baseia-se em criar uma arquitetura escalável e tolerante a falhas para redes de centro de dados, construída sobre OpenFlow. O próximo capítulo traz uma breve descrição de alguns trabalhos que tratam de roteamento e encaminhamento em redes de *data center*.

3 Trabalhos Relacionados

Recentemente, tem havido uma série de propostas de arquiteturas de rede, visando especificamente os centros de dados. O presente capítulo relaciona alguns dos trabalhos mais relevantes no contexto da pesquisa desenvolvida, trazendo as ideias chave das propostas e abordagens vinculadas aos objetivos relacionados no Capítulo 1. São apresentados diferentes trabalhos que abordam o encaminhamento e roteamento em redes de *data center*.

A Seção 3.1 apresenta as principais topologias utilizadas por trabalhos recentes relacionados a centros de dados de grande escala. A Seção 3.2 discute os trabalhos VL2 e PortLand, que apresentam propostas de encaminhamento de camada 2 em redes de *data center*; a Seção 3.3 aborda o trabalhos que utilizam conceitos de roteamento na origem de forma a aumentar a escalabilidade; finalmente, a Seção 3.4 mostra as contribuições deste trabalho ao estado da arte.

3.1 Novas topologias

Trabalhos propostos recentemente sugerem topologias alternativas para redes de *data center* de grande escala. Apresentaremos algumas das principais topologias que tiveram maior destaque, sendo utilizadas pelas principais arquiteturas de *data center* da literatura corrente.

3.1.1 Fat-trees

A diferença de preço entre equipamentos de prateleira (*commodity*) e equipamentos especializados levam a uma tendência de se utilizar muitos *switches* commodity ao invés de uma pequena quantidade de grandes *switches* de alto custo. Mais de cinquenta anos atrás, um problema similar levou Charles Clos a projetar uma rede de telefonia que tivesse alta largura de banda utilizando pequenos comutadores *commodity* (CLOS, 1953).

Baseadas na redes Clos, as *fat-trees* (AL-FARES; LOUKISSAS; VAHDAT, 2008) são árvores multi enraizadas, que possuem múltiplos caminhos entre seus nós. Geralmente *fat-trees* são construídas usando *switches* idênticos em todos os níveis, garantindo que elas sejam rearranjáveis, e não necessitando de *switches* com grande número de portas nos níveis mais altos da hierarquia da árvore. Por esses motivos, esse tipo de topologia é atrativa para a interconexão de grande escala (HAMADA; NAKASATO, 2005; LEISERSON,

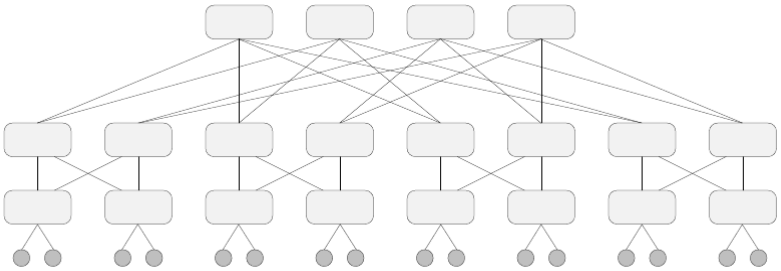


Figura 10: *Fat-tree* utilizando *switches* de 4 portas.

1985).

A organização de uma fat-tree é similar a de uma topologia de múltiplas camadas mostrada na Figura 2. De forma geral uma *fat-tree* de nível k é construída usando *switches* com k portas. As duas camadas mais inferiores são agrupadas em k pods, com cada pod consistindo de $k/2$ *switches* na camada inferior e $k/2$ *switches* na camada superior. Cada *switch* da camada inferior é ligado a todos $k/2$ *switches* da camada superior do pod, bem como a $k/2$ *hosts*. No nível superior existem $(k/2)^2$ *switches* de núcleo, que são ligados a um *switch* em cada pod. Dessa forma, em uma topologia fat-tree existem $k^2/4$ caminhos entre cada par de *hosts* em pods diferentes. Um exemplo de uma *fat-trees* com k=4 é mostrado na figura 10.

3.1.2 B-Cube

A arquitetura de rede BCube (GUO et al., 2009) utiliza a abordagem *server-centric* (centrada no servidor), o que significa que os servidores também participam do encaminhamento. Logo os servidores devem possuir mais de uma interface de rede, que tipicamente não é maior que cinco. A topologia BCube foi proposta para utilização em *data centers* modulares (MDC - *Modular Data Centers*), que são geralmente montados em contêineres, possuindo maior facilidade em sua migração para outras localidades geográficas. Para tal, a topologia BCube emprega múltiplas camadas de mini-*switches* para conectar os servidores. BCube oferece vários caminhos paralelos curtos entre qualquer par de servidores. Isto não só fornece largura de banda, mas também melhora a tolerância a falhas e balanceamento de carga.

A topologia BCube é definida de forma recursiva, sendo uma rede $BCube_n$ constituída por n redes do tipo $BCube_{n-1}$. O componente fundamental da topologia e uma rede $BCube_0$, composta n servidores conectados a um único *switch* de n portas. Para a construção de uma $BCube_1$ (Figura 11) utiliza-se n redes $BCube_0$ e n comutadores. Cada comutador é conectado a todas as redes $BCube_0$ através da conexão com um dos servidores de cada $BCube_0$. De forma mais geral, uma $BCube_k$ ($k >= 1$) é construída

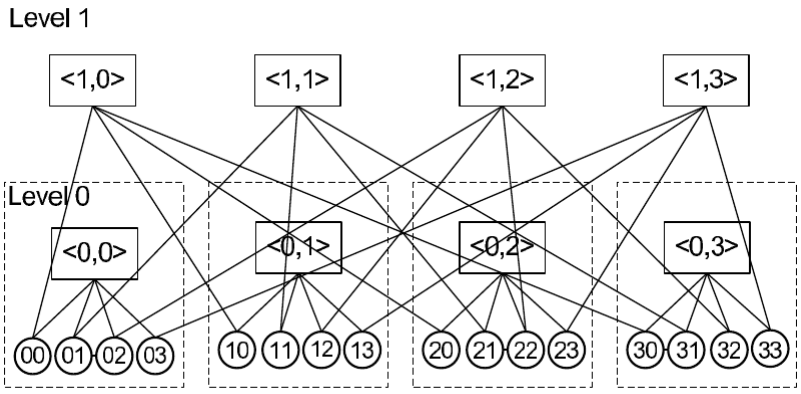


Figura 11: BCube de nível 1 com n=4. (GUO et al., 2009)

de n $BCube_{k-1}$ e n*k *switches* de n portas. Cada servidor em uma $BCube_k$ possui $k+1$ portas, numeradas de 0 a k. É fácil perceber que uma $BCube_k$ terá n*$k+1$ servidores e $k+1$ níveis de *switches*, com cada nível tendo n*k *switches* de n portas.

3.1.3 D-Cell

Da mesma forma que a BCube, a DCell (GUO et al., 2008) é definida recursivamente e utiliza encaminhamento pelos servidores e mini-*switches*. Devido à sua estrutura, DCell consegue ser escalável usando apenas mini-*switches*. Na prática, uma DCell de pequeno grau (por exemplo, 4) pode suportar milhões de servidores sem o uso de *switches* ou roteadores de alto desempenho. Em uma rede DCell não há nenhum ponto único de falha, sendo possível suportar várias falhas nos enlaces e servidores.

O componente fundamental da topologia é a rede $DCell_0$ constituída, assim como a $BCube_0$, por um comutador ligado a n servidores. Constrói-se uma $DCell_1$ utilizando $n+1$ redes $DCell_0$, na qual duas $DCell_0$ estão conectadas entre si através de um enlace formado por um de seus servidores. A $DCell_1$ está ilustrada na Figura 12.

Para a construção de uma $DCell_k$ são necessários $n+1$ redes $DCell_{k-1}$. Cada servidor em uma rede $DCell_k$ possui $k+1$ enlaces, sendo que em cada servidor, o primeiro enlace (enlace de nível 0) é conectado ao comutador da $DCell_0$ da qual ele faz parte, o segundo enlace conecta o servidor a um nó de uma mesma $DCell_1$, mas em uma $DCell_0$ vizinha. Genericamente, o enlace de nível i de um servidor o conecta a uma $DCell_{i-1}$ vizinha dentro de uma mesma $DCell_i$. O procedimento de construção é mais complexo que o da rede BCube, sendo executado a partir de um algoritmo de formação proposto por (GUO et al., 2008)

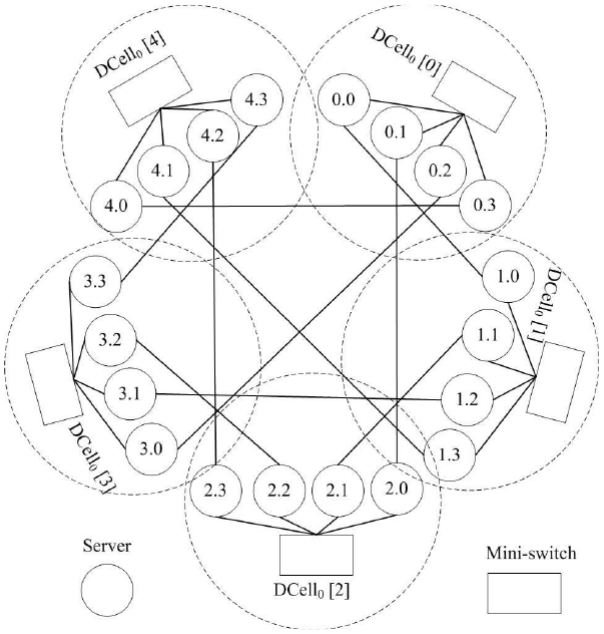


Figura 12: DCell de nível 1 com n=4. (GUO et al., 2008)

3.2 Projetos de redes de *Data Center*

Atualmente existe um grande esforço da comunidade (indústria e acadêmica) em busca do desenvolvimento de novas arquiteturas de *data center* voltadas para a resolução dos problemas citados nas seções anteriores. Nesse sentido, apresentamos arquiteturas que podem ser consideradas como as mais significativas e que representam o universo das pesquisas realizadas nesta área.

3.2.1 VL2

O VL2 (*Virtual Layer* 2) (GREENBERG et al., 2009) emprega uma camada 2 virtual que provê aos serviços do *data center* a ilusão de que todos os servidores associados a eles estão conectados através de um único *switch* de camada 2. Este trabalho apresenta uma arquitetura que busca prover agilidade na alocação de servidores através da introdução de uma camada de *software* (*shim layer*) na pilha de protocolos implementada pelos servidores, de tal forma a contornar as limitações impostas pelos dispositivos legados de rede. Logo, através de uma modificação do *software* (sistemas operacionais) utilizados nos servidores, o VL2 propõe uma reorganização nos papéis desempenhados tanto pelos servidores quanto pela rede.

3.2.1.1 Arquitetura

A Figura 13 apresenta a topologia Clos, que devido ao elevado número de conexões disponíveis entre qualquer par de *switches* de agregação, garante uma lenta degradação

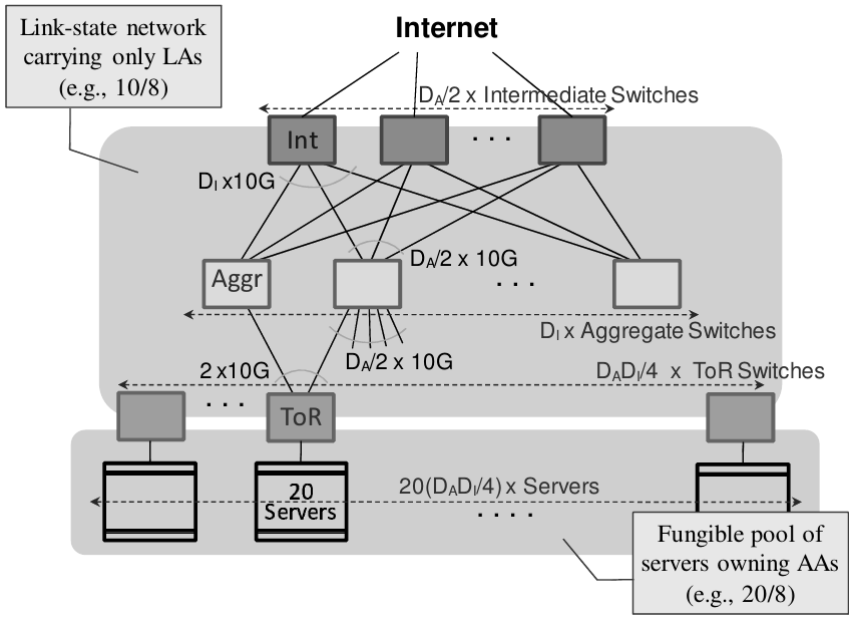


Figura 13: Topologia Clos usada pelo VL2. (GREENBERG et al., 2009)

da largura de banda nos cenários de falhas de enlace.

A rede é constituída por duas classes de endereços, um com significado topológico (LAs - *Locator Addresses*) e a outra de endereços planos de aplicação (AAs - *Application Addresses*). A infraestrutura de rede utiliza endereços LA, que são atribuídos para todos os *switches* e suas interfaces. Além disso, todos os *switches* executam um protocolo de roteamento baseado no estado do enlace para disseminar estes LAs, oferecendo uma visão global da topologia e possibilitando o encaminhamento de pacotes encapsulados em LAs através de caminhos mais curtos. Por outro lado, as aplicações utilizam AAs que permanecem inalterados, independentemente da maneira como os servidores migram no interior do *data center*. Para todo AA é associado o LA atribuído ao *switch* ToR no qual o servidor está conectado. Este mapeamento é mantido por um serviço de diretório do VL2.

3.2.1.2 Endereçamento e roteamento

A Figura 14 apresenta um exemplo de encaminhamento de pacotes através da estrutura do VL2. Basicamente, para rotear tráfego entre servidores identificados por endereços AA em uma rede que possui rotas formadas a partir de endereços LA, um agente VL2 executando em cada um dos servidores intercepta os pacotes originados e os encapsula em pacotes endereçados ao LA do *switch* ToR associado ao servidor de destino. Basicamente, o sucesso do VL2 está associado ao fato dos servidores acreditarem compartilhar uma única sub-rede IP, devido ao encapsulamento efetuado.

Como forma de contornar o tráfego *broadcast* gerado pelo protocolo ARP, na pri-

meira vez em que um servidor envia pacotes para um determinado endereço AA, a pilha de rede original do servidor gera uma requisição ARP e a envia para a rede. Neste instante, o agente VL2 presente no servidor intercepta a requisição ARP e a converte em uma pergunta *unicast* para o serviço de diretório do VL2. Este diretório, por sua vez, responde com o endereço LA do ToR de destino no qual os pacotes devem ser tunelados e o agente VL2 armazena este mapeamento, em um procedimento similar ao cache original do ARP, evitando novas perguntas ao serviço de diretório.

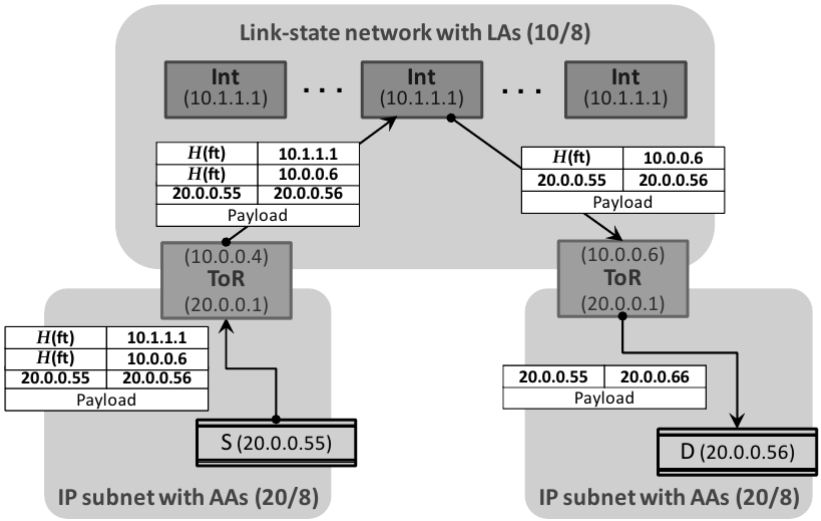


Figura 14: Encaminhamento de pacotes em uma rede VL2. (GREENBERG et al., 2009)

3.2.2 PortLand

Levando em consideração que em ambientes de *data center* a topologia base é conhecida e relativamente fixa, PortLand (MYSORE et al., 2009) propõe um conjunto de protocolos compatíveis com Ethernet para efetuar roteamento, encaminhamento e resolução de endereços, considerando a topologia da rede sendo *fat-tree*. Com base neste cenário, o PortLand propõe: (1) a existência de um serviço centralizado de gerenciamento da infraestrutura de rede (*Fabric Manager*); (2) a atribuição de pseudo endereços MAC (PMAC) para todos os nós finais, de forma a codificar suas posições na topologia; (3) a implantação de um serviço de *proxy* para contornar o tráfego *broadcast* inerente ao ARP e (4) a utilização de um protocolo para possibilitar que os *switches* descubram sua posição topológica na rede.

3.2.2.1 *Fabric Manager*

O PortLand utiliza um gerenciador de infraestrutura de rede centralizado, denominado *Fabric Manager*, responsável por manter estados relacionados à configuração da rede, auxiliar às requisições ARP, prover tolerância a falhas e operações de *multicast*. De

acordo com a especificação do PortLand, o *Fabric Manager* pode ser desenvolvido como um servidor conectado de forma redundante à estrutura do *data center* ou, ainda, ser executado em uma rede de controle separada.

3.2.3 Endereços PMAC (*Pseudo* MAC)

A base para um mecanismo de encaminhamento e roteamento eficiente no Portland, bem como o suporte à migração de máquinas virtuais, é a utilização dos endereços hierárquicos chamados Pseudo MAC (PMAC). O PortLand atribui para cada nó final um PMAC representando a localização do nó na topologia. Por exemplo, todos os nós finais localizados em um determinado pod compartilham um mesmo prefixo em seu PMAC. Entretanto, os nós finais permanecem inalterados, ou seja, eles acreditam ser identificados por seus endereços MAC reais (AMAC - *Actual* MAC). As requisições de ARP feitas pelos nós finais são respondidas com o PMAC do nó de destino. Sendo assim, todo processo de encaminhamento de pacotes ocorre através da utilização dos PMAC. Os *switches* de borda são responsáveis pelo mapeamento PMAC para AMAC e reescrita dos pacotes para manter a ilusão de endereços MAC inalterados no nó de destino.

Para todos os nós diretamente conectados, os *switches* de borda atribuem um PMAC de 48 bits sob o formato *pod.position.port.vmid*, onde *pod* possui 16 bits e refere-se ao número do pod onde os nós estão localizados, *position* possui 8 bits e indica a posição do *switch* dentro do pod e *port* possui 8 bits para representar a porta na qual o nó final está ligado ao *switches*. O campo *vmid* possui 16 bits e é utilizado para multiplexar máquinas virtuais em uma mesma máquina física.

Quando um *switch* de ingresso observa a existência de um novo endereço MAC, os pacotes com este endereço são desviados para o plano de controle do *switch*, que cria uma nova entrada na tabela de mapeamento e, na sequência, encaminha este novo mapeamento para o *Fabric Manager* para futuras resoluções, conforme ilustra a Figura 15.

Basicamente, o PortLand efetua a separação entre localizador/identificador de forma totalmente transparente aos nós finais e compatível com o *hardware* dos *switches* comoditizados disponíveis no mercado. Outra característica importante do PortLand refere-se a não utilização de técnicas de tunelamento para encaminhar os pacotes, sendo apenas necessário a reescrita de endereços PMAC/AMAC nas bordas do *data center*.

3.3 Roteamento na origem

Roteamento na origem é uma abordagem atraente que separa a lógica de roteamento de dados e fornece escalabilidade no plano de dados tornando os elementos de encaminhamento sem estado (JOHNSON; MALTZ, 1996). Em contraste com o encaminhamento convencional, os *switches* encaminham pacotes com base apenas nas informa-

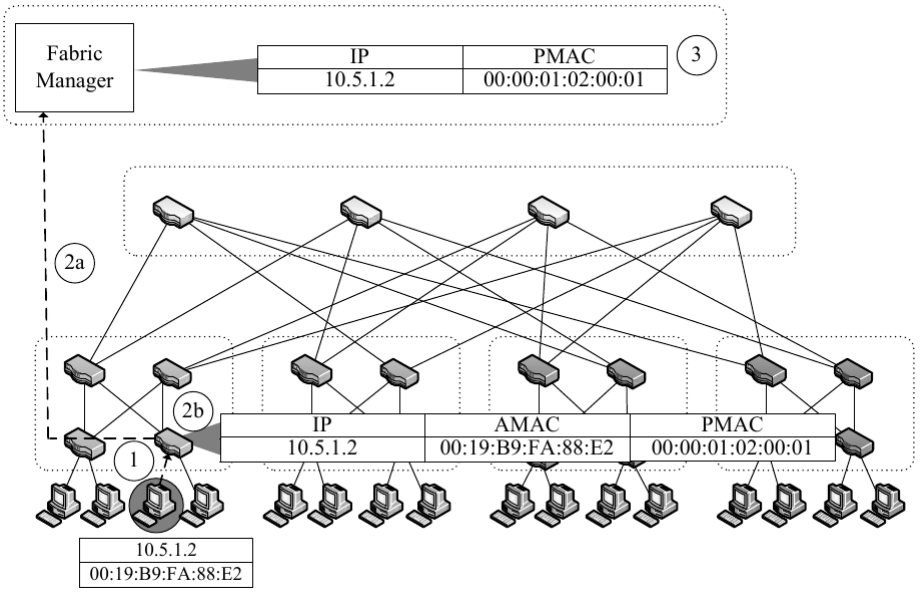


Figura 15: Mapeamento de MAC para PMAC. (MYSORE et al., 2009)

ções de roteamento transportadas nos cabeçalhos dos pacotes. Vamos agora avançar sobre um conjunto de trabalhos relevantes recentes para o escopo desta dissertação.

3.3.1 *Switching with in-Packet Bloom Filters* (SiBF)

SiBF (ROTHENBERG et al., 2010) é uma arquitetura de rede de centro de dados que oferece um novo serviço de encaminhamento baseado na codificação de rotas na origem em um filtro de *Bloom* nos pacotes (iBF - *in-packet Bloom filter*), no caso, adicionado aos campos de endereço do quadro MAC/Ethernet. O encaminhamento com SiBF assume a ideia de mover o estado da rede para os cabeçalhos dos pacotes, representando de forma compacta o caminho em um filtro de *Bloom* alocado nos campos MAC Ethernet.

3.3.1.1 Filtro de Bloom

Um filtro de Bloom é uma estrutura de dados que identifica se determinado elemento está ou não presente nessa estrutura. É implementado como um vetor de m bits, com n elementos inseridos por k funções de *hash* independentes. Cada elemento, ao ser inserido no filtro, passa por uma função de *hash* cuja ação consiste na atribuição do valor 1 ao bit na posição do vetor associada ao resultado do *hashing*. Para verificar se algum elemento está presente, os resultados de todas as k funções *hash* devem possuir o valor 1 no vetor. Basta um único dos bits anteriores apresentar o valor zero para concluir que o elemento não está presente. Neste caso, não há possibilidade da ocorrência de falsos negativos. Um falso positivo ocorre quando todos os resultados das k funções de *hash* em um elemento que não foi inserido apresentam o valor 1.

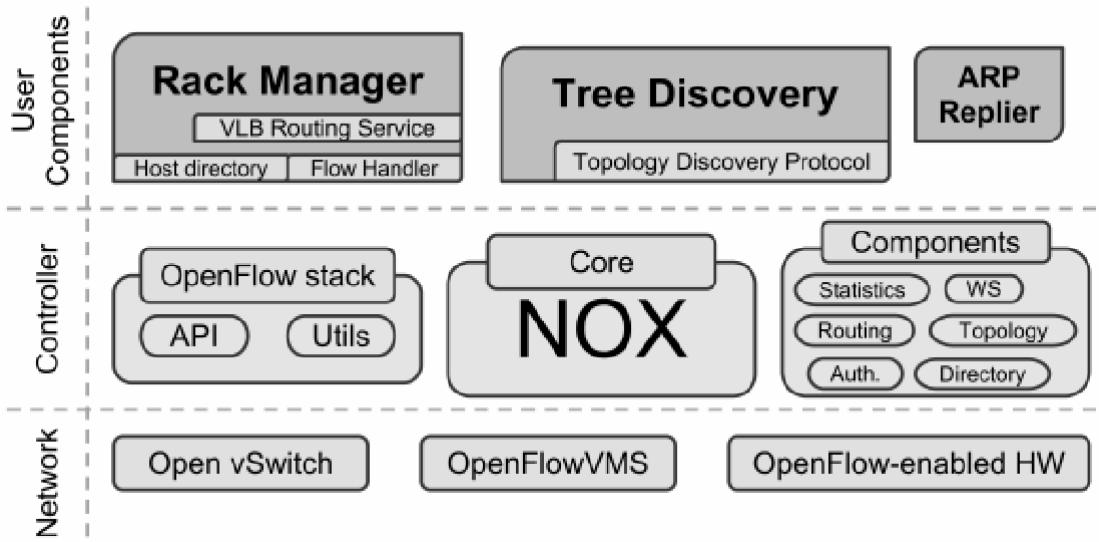


Figura 16: Arquitetura do SiBF. (ROTHENBERG et al., 2010)

Após a descoberta da topologia é instalada na tabela de encaminhamento dos *switches* uma entrada de fluxo correspondente a cada uma de suas portas de saída. Cada uma destas entradas corresponde a um filtro de *Bloom* codificado com a identificação do vizinho detectado. Quando um pacote com iBF chega ao *switch*, a máscara de bits é comparada com as entradas na tabela; se uma entrada for encontrada, o pacote é enviado à porta correspondente. Um falso positivo ocasiona o envio do pacote para duas (ou mais) interfaces de saída.

Uma opção para contornar a ocorrência de falsos positivos é a realização do *multicast*. Neste caso, o pacote será encaminhado pelo caminho certo e por mais uma porta e, no próximo *switch*, com alta probabilidade, será descartado ao carecer de uma entrada válida. Outra opção consiste no encaminhamento *unicast*. Nesta opção, se a escolha por um dos caminhos não for a correta, o pacote será encaminhado pelo próximo *switch* (que não terá uma entrada válida para encaminhar aquele pacote) ao controlador, permitindo, neste caso, que uma nova rota seja definida. No entanto, como o controlador possui a visão global da topologia e existem múltiplos caminhos (utiliza-se a *fat-tree*), a opção adotada no trabalho (MACAPUNA et al., 2010) consiste na eliminação dos caminhos sujeitos a falsos positivos.

3.3.1.2 Encaminhamento

O encaminhamento em um *fat-tree* de 3 camadas fica bastante simplificado, pois qualquer rota entre dois ToRs tem uma trajetória ascendente em direção a um *switch* Core e, em seguida, uma trajetória descendente em direção ao ToR destino, ambas passando por *switches* intermediários (Aggr). Esta abordagem permite enviar pacotes por

rotas diferentes (balanceamento de carga), além de permitir a seleção da rota menos congestionada (engenharia de tráfego). A especificação da rota na origem é baseada na utilização do filtro de Bloom contendo apenas três elementos identificadores de *switches* ($Core$, $Aggr_{desce}$, $ToR_{destino}$). O identificador utilizado no trabalho corresponde ao endereço MAC do *switch*. O esquema básico adotado para operacionalização do filtro de Bloom consiste na programação do ToR para adicionar nos campos MAC_{src} e MAC_{dst} do cabeçalho do quadro Ethernet o filtro de *Bloom* contendo a rota. Na sequência, o ToR encaminha o quadro para o Aggr de subida. Deve ser ressaltado que não há necessidade de incluir o identificador do Aggr de subida no filtro de *Bloom* já que esta informação é implícita ao ToR. Nos próximos saltos, apenas três encaminhamentos são realizados utilizando o iBF, conforme exemplificado na Figura 17.

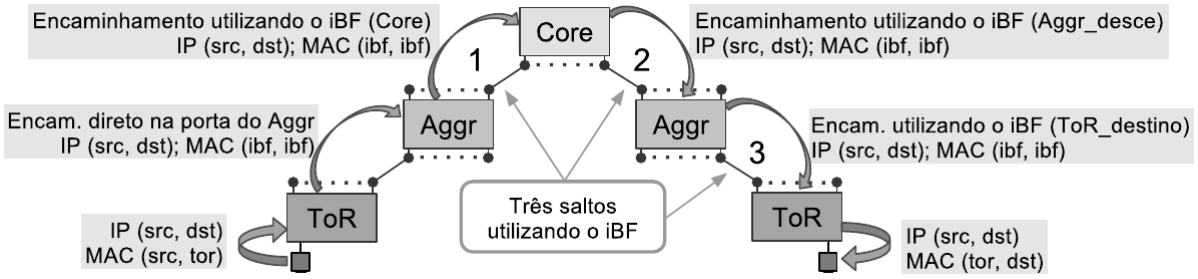


Figura 17: Encaminhamento utilizando o SiBF. (ROTHENBERG et al., 2010)

3.3.1.3 *Rack Manager*

O *Rack Manager* (RM) atua como um controlador de *switches* e a sua implementação é feita sobre um controlador NOX. O RM é responsável pelo gerenciamento dos novos fluxos que chegam ao *switch* e a respectiva configuração destes fluxos. O Rack Manager mantém uma base de dados em cache com informações dos servidores descobertos em cada ToR e um mapeamento entre IP, MAC e porta do ToR no qual estes servidores estão anexados. Para realizar o encaminhamento com o iBF e fornecer os serviços de topologia e diretório, são utilizados alguns componentes adicionais para o NOX (Figura 16).

***Tree Discovery*:** Módulo de inferência de topologia que utiliza uma extensão do LLDP (*Link Layer Discovery Protocol*) para divulgar a identificação de cada *switch*. Esta informação é propagada entre os vizinhos, onde cada um descobre qual é o seu papel na topologia (ToR, Aggr ou Core).

***ARP Replier*:** Possui a tarefa de registrar e responder as requisições ARP que chegam aos *switches* de borda, eliminando o tráfego *broadcast* gerado pelo protocolo ARP.

***Topology Service* (TS) e *Directory Service* (DS):** Oferecem métodos de inserção e recuperação das informações na base de dados. O TS e o DS são responsáveis

apenas pela atualização da base local (*cache*) do RM. Desta forma, tem-se uma base de dados logicamente centralizada (mas distribuída fisicamente) e replicada em cada RM.

O *Rack Manager* interage periodicamente com o *Topology Service* (algoritmo de inferência de topologia) e o *Directory Service* para obter a topologia completa e as informações globais de diretório. Desta forma, o RM constrói uma matriz com os candidatos iBFs livres de falsos positivos para cada caminho entre qualquer combinação de pares de ToRs.

3.3.2 *Slick Packets*

Slick Packets (NGUYEN et al., 2011) é uma proposta que, embora não seja destinada especificamente à redes de *data center*, serviu de inspiração para o projeto de recuperação de falhas proposto por este trabalho.

O objetivo deste trabalho é alcançar a reação rápida de falhas com rotas alternativas embutidos dentro da rede, e a flexibilidade de rotas inseridas nas bordas. Para cumprir esta meta, em vez de especificar um único caminho para o destino, o cabeçalho do pacote contém um grafo acíclico direcionado, o Subgrafo de Encaminhamento (FS - *Forwarding Subgraph*). Cada roteador ao longo do caminho do pacote pode optar por enviá-lo ao longo de qualquer um dos *links* de saída a partir do nó atual no FS (opcionalmente preferindo um caminho marcado como o principal). Esta abordagem permite que os pacotes desviem de falhas na rede, mantendo a flexibilidade de controle de rota de origem.

A Figura 18 ilustra a concepção do *Slick Packets*. O *host* de origem s envia um pacote para um destino d. Com posse do mapa da topologia da rede, o nó escolhe o subgrafo (FS), como mostrado na figura, e designa (s, $R1$, $R2$, $R5$, d) como o caminho primário. Note que o FS proporciona a cada nó no caminho primário caminhos alternativos suficientes para que, se uma ligação no caminho principal falhar, o pacote pode ser reencaminhado para o destino. Em seguida, s constrói um pacote de dados com o subgrafo incorporado no cabeçalho do pacote e o encaminha para o primeiro *hop* $R1$. Neste, o pacote é encaminhado para o próximo salto no caminho primário ($R2$). Suponhamos agora que em $R2$ o enlace para o próximo salto do caminho primário $R5$ tenha falhado. Neste caso, o pacote será encaminhado de $R2$ para $R4$, o próximo salto no seu caminho alternativo, em seguida o pacote segue para $R5$ e finalmente d.

3.4 Contribuições para o estado da arte

O presente trabalho destina-se a construir uma arquitetura que utiliza roteamento na origem, onde as fontes especificam um caminho no cabeçalho do pacote, contrastando com o modelo tradicional de roteamento distribuído. O roteamento na origem combinado

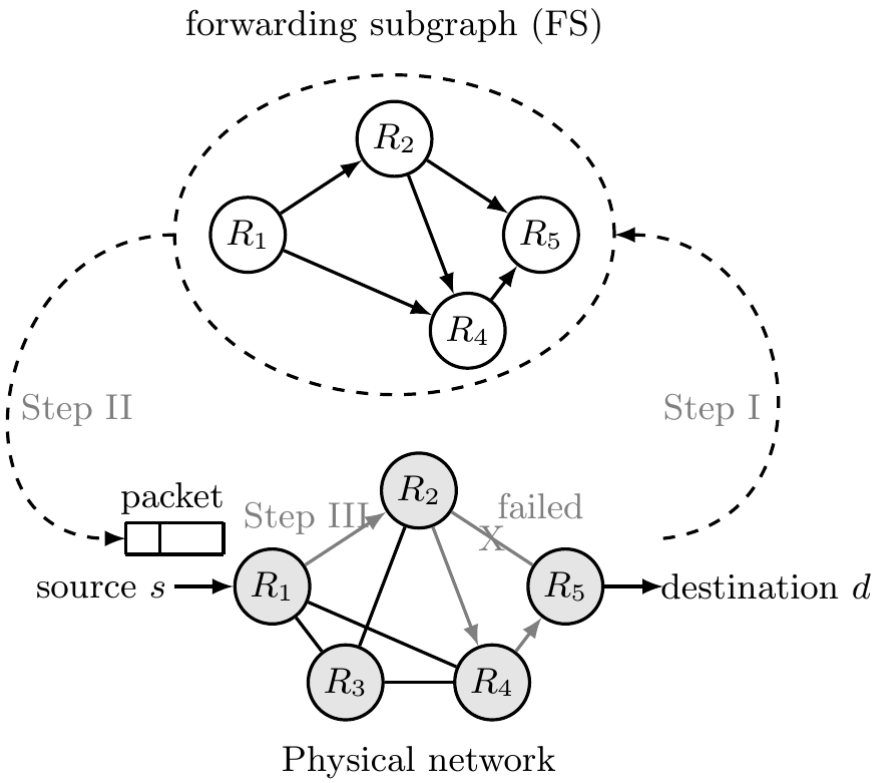


Figura 18: Visão Geral do *Slick Packets*. (NGUYEN et al., 2011)

com o modelo de controle centralizado do SDN/OpenFlow permite que a proposta seja empregada facilmente em diferentes topologias.

Comparado com os trabalhos relacionados, este trabalho traz diversos avanços. Ao contrário do VL2, por exemplo, nossa proposta evita modificações nos *hosts* e sobrecarga com cabeçalhos adicionais. Além disso, empregamos caminhos pré-computados que leva à redução da sobrecarga na instalação dos caminhos e na recuperação de falhas. Ainda, a abordagem apresentada neste trabalho não usa estruturas de dados probabilísticas, como os *Bloom Filters*, evitando falsos positivos, que podem levar a ter mais de um candidato para encaminhar os pacotes e consequentemente reduzir a multiplicidade de caminhos.

A robustez da topologia do *data center* é um aspecto crítico (COUTO; CAMPISTA; COSTA, 2012), entretanto podemos notar que existe uma escassez de dados empíricos sobre falhas nas redes. Em relação a este aspecto, este trabalho apresenta importantes contribuições, propondo e avaliando dois mecanismos de recuperação de falhas. O primeiro utilizando controlador centralizado para alterar as regras que definem as rotas na origem. Posteriormente, inspirados pelo *Slick Packets*, implementamos e avaliamos a utilização de caminhos alternativos nos cabeçalhos dos pacotes, tornando a arquitetura capaz de se recuperar de falhas sem a intervenção do controlador.

4 Caminhos codificados em redes de *Data Center*

Neste capítulo serão apresentados os princípios de projeto e detalhes de implementação de uma arquitetura baseada em redes OpenFlow que emprega rotas na origem para simplificar o encaminhamento de pacotes e aumentar a escalabilidade na rede. Será descrito também como se dá a recuperação de falhas na rede, empregando o controlador para trocar os caminhos usados pelos fluxos. A proposta é implementada e validada em um ambiente de prototipação baseado em OpenFlow considerando uma arquitetura na qual um controlador de rede logicamente centralizado é responsável por operar de forma transparente o *data center*.

4.1 Visão Geral

Este trabalho contribui com um conjunto de mecanismos de encaminhamento, roteamento e tolerância a falhas compatíveis com o protocolo Ethernet com o objetivo de atender aos requisitos de redes de *data center*. Empregando os princípios de SDN, propomos um esquema que emprega um controlador centralizado que define regras nos switches de origem para embutir informações de caminho no cabeçalho dos pacotes na forma de caminhos codificados (EP – Encoded Paths). O EP especifica um conjunto de nós no *data center* através do qual os *switches* intermediários devem encaminhar os pacotes.

4.1.1 Princípios de projeto

Adotamos os seguintes princípios como base para a idealização do projeto da arquitetura proposta:

- **Separação da computação de rotas do tratamento de falhas:** A arquitetura proposta é baseada na separação de cálculo das rotas (na topologia implantada) do tratamento de falhas (quando ocorrem mudanças no estado dos enlaces) (CAESAR et al., 2010). Assim, a idéia central é calcular cada caminho em potencial de forma antecipada, com base unicamente na topologia implantada. Isto é, o roteamento deve ignorar todas as mudanças de estado e apenas recalcular caminhos quando ocorrerem mudanças de topologia. Uma vez que estas mudanças na topologia são raras e muitas vezes conhecidas com

antecedência, o cálculo de rotas pode ser feito de forma centralizada. Esta flexibilidade, tanto na duração quanto na localização do cálculo, permite uma grande variedade de critérios na computação de caminhos, por exemplo a preferência por rotas disjuntas.

- **Controle centralizado:** Um *data center* é administrado por uma entidade única. Isso nos leva a adotar um controlador de rede logicamente centralizado (NC - *Network Controller*) que toma todas as decisões no *data center*. Basicamente, o NC recebe um mapa da rede (ver detalhes na Seção. 4.2) para (i) calcular todas as rotas entre cada par de *hosts*, e (ii) instalar as regras de encaminhamento nos *switches* intermediários com base unicamente em seus vizinhos.

- **Roteamento na origem:** Como o controlador conhece a topologia, pode-se retirar dos *switches* a tarefa de tomar decisões de roteamento. Isso nos leva a usar o roteamento na origem, aumentando a escalabilidade do plano de dados.

  Roteamento de origem é uma técnica em que o emissor de um pacote determina a sequência completa de nós através dos quais encaminhar o pacote. Ou seja, o emissor enumera explicitamente a rota no cabeçalho do pacote, identificando para cada *hop* no caminho do pacote o endereço do próximo nó ao qual o pacote deve ser transmitido.

  Assim, propomos que o controlador permita que o *switch* de origem insira as informações do caminho no cabeçalho dos pacotes em forma de **caminho codificados (EP - *Encoded Path*)**. O EP especifica um conjunto de nós de dentro do *data center* através do qual os *switches* intermediários devem encaminhar os pacotes. No caso de um evento de falha (link ou *switch*), o mecanismo de recuperação consiste em modificar o caminho codificado embutido nos pacotes, permitindo usar outros caminhos disponíveis, sendo que os caminhos candidatos estão previamente calculados.

- **Separação do localizador:** O encaminhamento proposto não usa endereços IP para localizar os *hosts* dentro da DCN. Desta forma, ao contrário da atribuição hierárquica tradicional de endereços IP, não há restrições sobre como os endereços serão alocados. Em essência, nós separamos o localizador do identificador de modo que seja transparente para os *hosts* e compatível com o *hardware* existente. Além disso, esta abordagem não introduz cabeçalhos adicionais, tais como MAC-in-MAC ou IP-in-IP (GREENBERG et al., 2009; GREENBERG et al., 2008).

A Figura 19 mostra um exemplo para ilustrar o projeto proposto. Suponha que a origem s deseja enviar um pacote para um destino d. O pacote não encontra uma

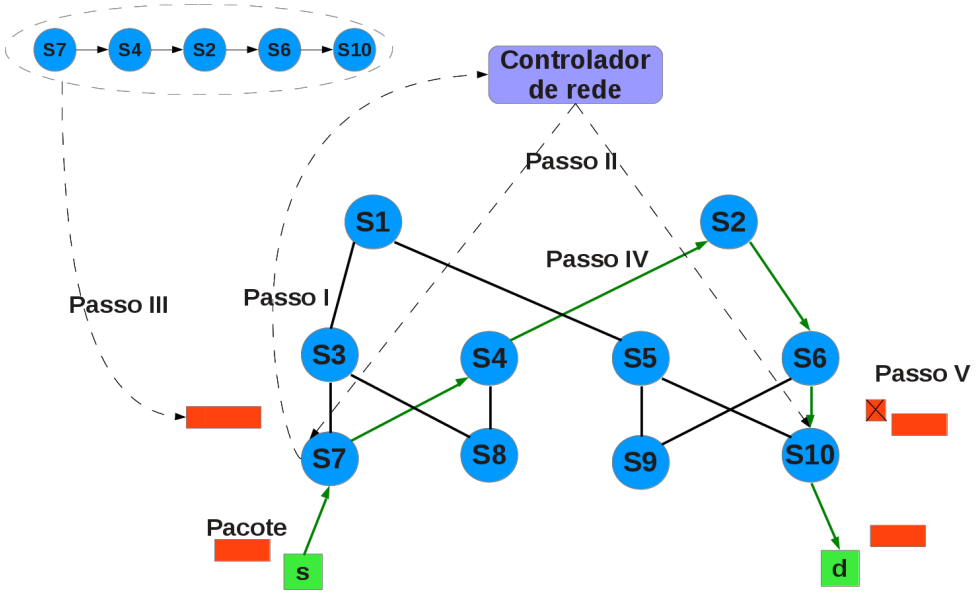


Figura 19: Visão geral do projeto.

regra na tabela fluxo no *switch* ToR de origem $S7$ e é encaminhado para o NC (passo I). O NC seleciona o caminho $S7 - S4 - S2 - S6 - S10$ e instala uma regra nos ToR de origem e de destino (passo II). A regra no ToR de origem $S7$ instrui o *switch* para (i) incorporar o caminho codificado (EP) selecionado no cabeçalho dos pacotes (passo III), e (ii) encaminhar o pacote para $S4$. Nos *switches* intermediários, $S4$, $S2$ e $S6$, todas as decisões de encaminhamento são baseadas no EP (passo IV). Finalmente, a regra no ToR de destino $S10$ remove o EP e entrega o pacote para o destino d (passo V).

A vantagem da abordagem baseada em roteamento na origem é que a tarefa de encaminhamento de cada *switch* torna-se trivial, uma vez que requer apenas o conhecimento local dos seus vizinhos, em vez de uma entrada na tabela de encaminhamento para cada destino potencial. Além disso, como o primeiro comutador especifica o EP, os múltiplos caminhos disponíveis podem ser explorados. Como os caminhos são pré-computados, quando uma falha ocorrer em qualquer enlace da topologia, um outro caminho com nós disjuntos pode ser rapidamente selecionado, apenas embutindo um caminho diferente no cabeçalho dos pacotes, sem a necessidade de algoritmos distribuídos de convergência pela rede.

É importante destacar que o mecanismo de encaminhamento é independente da topologia empregada no *data center*. É possível aplicar o encaminhamento com EP por exemplo às *fat-trees*, ou mesmo topologias *server-centric* como BCube (GUO et al., 2009) e DCell (GUO et al., 2008).

A seguir, descrevemos os detalhes dos mecanismos subjacentes.

4.2 Descoberta da topologia e cálculo das rotas

Para se construir rotas na origem é preciso que o NC, no momento da inicialização, conheça a topologia da rede sem intervenção manual. Para isso, usamos mensagens LLDP (*Link Layer Discovery Protocol*) para a inferência de topologia. Pacotes LLDP são enviados de cada interface dos *switches* e depois, de acordo com os pacotes recebidos nos *switches* vizinhos as ligações são descobertas.

O módulo de cálculo de rotas pode ser configurado para calcular todas as rotas entre todos os pares de *hosts*, ou um número predeterminado k de caminhos. Este parâmetro pode ser definido pelo administrador com base no tamanho da rede. Para um melhor desempenho, as rotas podem ser selecionadas de acordo com vários critérios, por exemplo com base em caminhos disjuntos (MAHAPATRA; YUAN; NIENABER, 2012), ou selecionando caminhos aleatoriamente a partir de uma distribuição de probabilidade (YUAN et al., 2007).

4.3 Encaminhamento com EP

O encaminhamento com caminhos codificados simplifica a função dos *switches*. Tradicionalmente, o encaminhamento dos pacotes é baseado no endereço de destino. Nos comutadores Ethernet e roteadores IP, os pacotes são encaminhados com base nos endereços MAC e IP, respectivamente.

O encaminhamento EP é muito mais simples. Ao invés de se basear nos endereços MAC e IP, o cabeçalho do pacote leva a sequência de *switches* por onde o pacote deve ser encaminhado. A Figura 20 mostra como o mecanismo de encaminhamento funciona. Neste caso, cada pacote carrega o cabeçalho EP, uma sequência de segmentos, um para cada *switch* no caminho do pacote, onde cada segmento correspondente a um comutador S_i contém o código que representa o próximo nó do caminho (I). O *switch* extrai o código contido no primeiro segmento(mais a esquerda) e o usa como chave de busca na tabela de fluxos para se obter a porta de saída (II). Em seguida, o EP é deslocado para a esquerda (III) e o pacote encaminhado para o próximo nó do caminho(IV).

O projeto proposto simplifica significativamente as funcionalidades dos *switches*, pois estes não estão envolvidos no roteamento. Suas tabelas de encaminhamento têm tamanho fixo, e seu comprimento é pequeno, uma vez que um nó normalmente tem no máximo dez dos vizinhos (GUO et al., 2010). A título de comparação, uma tabela de encaminhamento baseado no MAC de destino (ou IP) podem ter milhares de entradas e seu tamanho aumenta à medida que a rede se expande (linearmente com o número de dispositivos de rede ou no pior caso com o número total de *end-hosts* ou VMs).

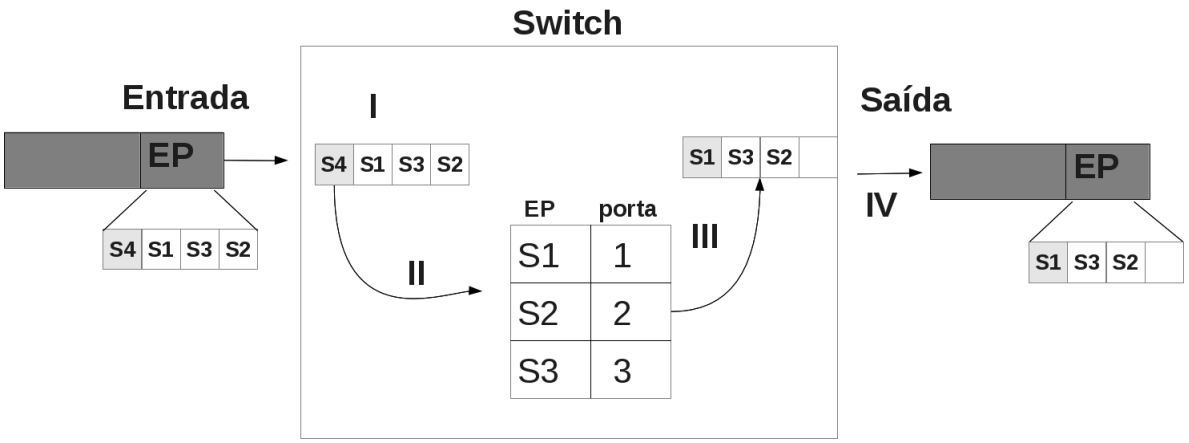


Figura 20: Encaminhamento com EP.

4.4 Implementação

A implementação do mecanismo de encaminhamento com EP foi baseado em *switches* OpenFlow (MCKEOWN et al., 2008), e o controlador de rede (NC) foi implementado como uma aplicação no NOX (GUDE et al., 2008).

Quando um pacote chega ao ToR e não existe uma regra para ele na tabela de fluxo, este é enviado para o NC, que seleciona algum dos caminhos previamente calculados entre a origem e o destino e instala regras nos *switches* de borda. No ToR de origem, a regra adiciona o EP ao cabeçalho do pacote. Já no ToR de destino, a regra remove o EP e entrega o pacote ao *host*. As entradas de fluxo nos *switches* de borda são baseadas nos endereço IP de destino (além de, opcionalmente VLAN ou identificadores dos *host* final).

O conceito em si é independente da localização do cabeçalho no pacote. Ele poderia estar em um cabeçalho adicional entre as camadas IP e MAC, em uma opção IP, ou em novo cabeçalho de uma próxima geração de protocolo da Internet. Nosso objetivo na concepção do formato de codificação é garantir o encaminhamento simples de dados e a possibilidade de ser implementado no *hardware* utilizado, no caso, *switches* OpenFlow. No entanto, optamos por utilizar os cabeçalhos existentes para evitar a sobrecarga adicional e simplificar a implementação.

Optou-se por colocar o cabeçalho EP nos campos $VLAN_{VID}$ e MAC Ethernet (vide Figura 21). O campo $VLAN_{VID}$, que possui 12 bits, foi usado como um segmento, e os 96 bits dos campos MAC Ethernet foram divididos em 12 segmentos de 8 bits. Dessa forma, o encaminhamento com EP suporta redes cujo caminho mais longo seja de até 13 saltos, e cada *switch* pode ter até 256 vizinhos (2^8), um *fan-out* mais que adequado considerando o número tipico de portas de switches comerciais.

Outras configurações seriam perfeitamente válidas.

O encaminhamento nos *switches* intermediários é baseado nas regras fixas instala-

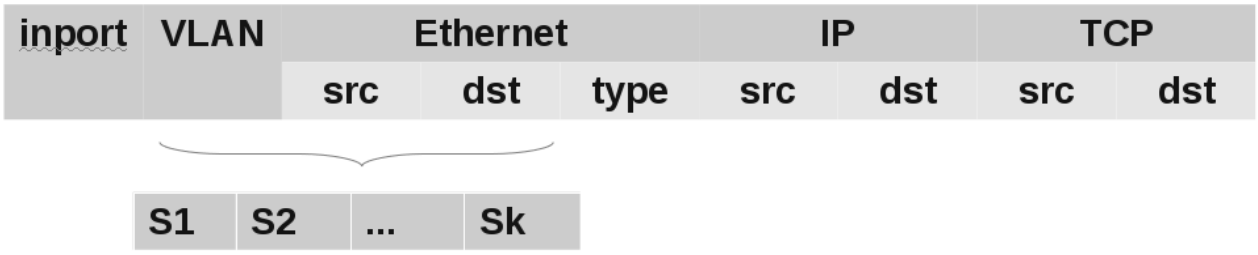


Figura 21: Posição do EP no cabeçalho.

das pelo NC durante a inicialização. Estas regras, com base nos segmentos do EP, indicam para qual dos vizinhos o pacote deve ser enviado. O *software* do *switch* OpenFlow foi modificado para que, a cada salto, o EP seja deslocado de um segmento para a esquerda, mantendo o proximo segmento a ser utilizado no campo $VLAN_{VID}$. Assim, os *switches* intermediários fazem o *match* sempre com o mesmo campo ($VLAN_{VID}$).

É importante destacar que os códigos transportados pelo EP são apenas localmente exclusivos para cada nó, o que é crítico para conseguir uma codificação compacta, porque o número médio de vizinhos de um *switch* de uma rede do mundo real é, tipicamente, muito menor do que o número total de *switches* da rede. Assim, é possível reduzir o tamanho da codificação de forma significativa em comparação à utilização de códigos globais.

Para melhorar o tempo de instalação dos fluxos fim-a-fim, o NC instala nos ToRs, simultaneamente, as regras para o caminho de ida e de volta, para que a resposta possa passar diretamente através do plano de dados sem envolver o NC.

No encaminhamento com EP, o endereço IP é usado apenas para identificar os *hosts* (ou VMs) dentro do *data center*. Desta forma, ao contrário do modelo hierárquico tradicional de endereços IP, não há restrições sobre como endereços são atribuídos. Isto faz com que o endereço IP não seja significativo para roteamento de pacotes dentro da rede. Em essência, nós estamos separando a localização do *host* do seu identificador de forma transparente. Além disso, essa abordagem não introduz cabeçalhos adicionais, tais como MAC-in-MAC ou IP-em-IP (GREENBERG et al., 2009).

Numa rede OpenFlow, o serviço de encaminhamento EP pode ser oferecido só para uma fração do tráfego ou das aplicações (ou VMs). Nesse cenário, o restante do tráfego pode ser tratado de forma legada, seguindo um modelo de rede híbrida, ou conforme determinado por outro controlador operando em paralelo, por exemplo mediado por um *proxy* virtualizador como o FlowVisor (SHERWOOD et al., 2009).

4.5 Resolução de requisições ARP

O protocolo de resolução de endereços (*Address Resolution Protocol*-ARP) é usado para encontrar um endereço da camada de enlace (Ethernet, por exemplo) a partir do endereço da camada de rede (como um endereço IP). O Ethernet por padrão transmite as mensagens ARP por *broadcast* para todos os *hosts* no mesmo domínio de camada 2. Aproveitamos o NC para reduzir a sobrecarga de transmissão, realizando a resolução de endereços de forma centralizada. Quando um *switch* ToR recebe um pedido ARP para um IP, encaminha o pedido ao NC, que mantém uma tabela de resolução ARP, e verifica se existe uma entrada para o endereço IP de destino. Caso exista, é criada uma resposta ARP e esta é enviada ao *host* de origem. Se o NC não tem o endereço MAC para o IP solicitado, a requisição ARP será encaminhada para todos os *hosts* finais para estabelecer o mapeamento, e as respostas serão encaminhadas novamente ao controlador. Este por sua vez, adicionará uma entrada em sua tabela de resolução e encaminhará a resposta ao *host* solicitante .

4.6 Balanceamento de carga

Projetar redes de *data center* utilizando topologias ricas em múltiplos caminhos, como as *fat-trees*, por si só, não resolve a exigência de alta largura de banda. Estas redes requerem mecanismos de balanceamento de carga para melhor utilizar os múltiplos caminhos para fornecer alta largura de banda fim-a-fim.

O balanceamento de carga pode ser realizado sem informação de estado e sem sobrecarga utilizando o mecanismo de ***oblivious routing*** (YUAN et al., 2007). Neste mecanismo, o caminho entre os nós de comunicação de origem (s) e destino (d) é selecionado aleatoriamente a partir de uma distribuição de probabilidade sobre todos os caminhos de s para d. Nós seguimos essa abordagem e deixamos o NC selecionar aleatoriamente, segundo uma distribuição uniforme, um entre os caminhos pré-computados.

Em tempo de execução, o NC pode decidir empregar entradas adicionais nas tabela de fluxo para sobrepor o encaminhamento inicial para fluxos individuais. Por exemplo, sabendo que certos *hosts*, ou determinadas portas TCP são usadas para trocar grandes volumes de dados, como *backups*, regras específicas para esses fluxos podem ser instaladas, evitando o caminho padrão, potencialmente usando conhecimento adicional sobre os requisitos da aplicação e do estado da rede.

4.7 Roteamento tolerante a falhas

Dada a topologia e a capacidade de encaminhamento com base nos EPs, um aspecto importante é a tolerância a falhas. Propomos um mecanismo de restauração de

caminho de forma centralizada, empregando o NC para trocar o caminhos dos fluxos quando uma falha ocorre na rede. Quando a sinalização da falha é recebida dos *switches*, o NC busca nos caminhos pré-calculados uma rota alternativa livre de falhas, e altera os caminhos codificados nos *switches* de borda.

No presente trabalho, o foco concentra-se na recuperação de falhas na rede após estas serem detectadas, independentemente do método de detecção a ser utilizado. Abordagens de detecção de falhas possíveis incluem mecanismos em nível de caminho (SUCHARA et al., 2011; FU; LE; ZHU, 2012) ou utilização de mensagens de descoberta LLDP. Mecanismos de detecção diferentes irão produzir tempos de recuperação diferentes. Para fins de avaliação, assumimos um evento `port-down` recebido pelo controlador que irá acionar um mecanismo de recuperação de falha, tal como descrito a seguir.

O processo de recuperação de falhas pode ser descrito usando o exemplo ilustrado na Figura 22. O *switch* detecta uma falha no enlace (passo I) e informa ao controlador NC sobre esta falha (passo II). O NC atualiza a matriz de conectividade, que mantém o estado dos enlaces da rede (passo III) e identifica os fluxos instalados que passam através do enlace afetado. O NC então escolhe novos caminhos disponíveis para os fluxos afetados e envia mensagens de `flow-mod` (passo IV) para os ToRs correspondentes modificarem o EP, para que o pacote seja roteado por um caminho alternativo.

Protocolos de roteamento tradicionais, como o OSPF, exigem a comunicação entre todos os n *switches* com $O(n^2)$ mensagens de rede. Nossa proposta requer uma mensagem do *switch* que detecta a falha para o controlador, e uma mensagem para cada ToR que encaminha pacotes para as rotas descontinuadas, resultando em uma comunicação da ordem de $O(t)$, onde t é o número de ToRs na rede.

Como alternativa, o NC poderia instalar duas regras em um *switch* ToR, cada uma especificando caminhos disjuntos com prioridades diferentes. Se o caminho de alta prioridade é interrompido, o NC (ou o próprio *switch*) pode simplesmente remover (ou diminuir a prioridade) essa entrada e os pacotes do fluxo corrente irão começar a correspondência com a entrada EP de *backup*. Explorando funcionalidades introduzidas nas novas versões do OpenFlow, tais como grupos de *fast failover*, o próprio *switch* pode mudar o comportamento de encaminhamento para os EPs de *backup* sem a necessidade de uma viagem de volta para o controlador. Estas otimizações são objeto de trabalhos futuros.

Falhas no controlador ou na ligação entre os *switches* e controladores podem ocorrer. Este tipo de problema foi abordado considerando um controlador logicamente centralizado, mas fisicamente distribuído (TOOTOONCHIAN; GANJALI, 2010). A resiliência do plano de controle está fora do escopo deste trabalho e pode ser abordada em implementações de protótipos futuros baseadas na versão 1.2 do protocolo OpenFlow que introduz suporte a múltiplos controladores em modo ativo e *backup*. Esses trabalhos devem se apoiar nas as melhores práticas da literatura (ex. (KOPONEN et al., 2010)) e

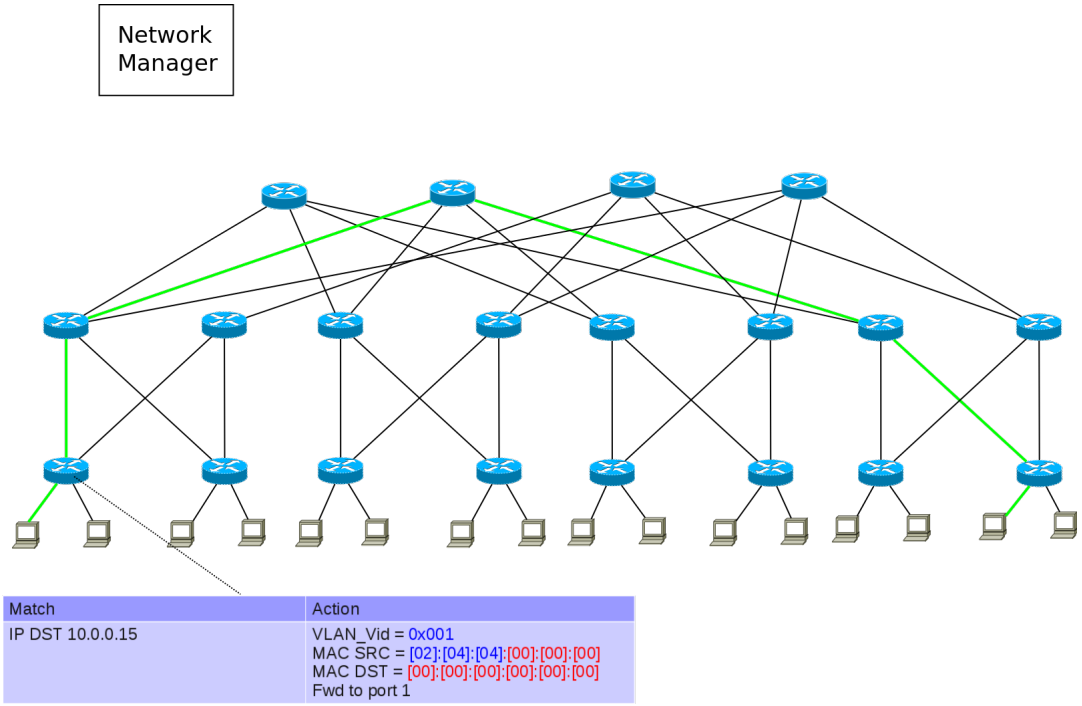


Figura 22: Recuperação de falhas. Após a falha, em vermelho, o caminho codificado no switch de origem será alterado pelo controlador.

os subjacentes princípios de sistemas distribuídos assim como nas soluções disponíveis de bases de dados NoSQL que tem sido sugeridas (MACAPUNA et al., 2010) como boas candidatas para distribuição do estado critico dos controladores OpenFlow.

4.8 Avaliação

Avaliamos a abordagem com caminhos codificados em termos do tamanho de codificação dos cabeçalhos, tempo de recuperação de falhas, desempenho de encaminhamento, balanceamento de carga e análise de requisitos de estado para redes *fat-tree*. A plataforma de testes foi construída sobre o Mininet (LANTZ; HELLER; MCKEOWN, 2010). Os *switches* se conectam a um controlador OpenFlow na mesma máquina. A versão do protocolo OpenFlow usado na implementação é a 1.0.

4.8.1 Tamanhos dos cabeçalhos

Começamos avaliando o tamanho dos caminhos codificados, usando o formato de codificação apresentado na Seção 4.4. Na nossa proposta, o tamanho do cabeçalho depende do diâmetro da topologia e do número de portas de saída dos *switches*. Assim, considerando uma rede com diâmetro d e cada *switch* com p portas (e $p*$ a próxima

potência de 2 superior a p), o cabeçalho EP terá d segmentos, cada um com o tamanho de $\log_2(p)$ bits, levando à seguinte equação para o tamanho do cabeçalho em bits:

$$S = d * (\log_2(p*)) \tag{4.1}$$

A Tabela 1 mostra o tamanho em bits do cabeçalho com caminhos codificados para algumas topologias, variando o diâmetro d e número de portas por swtich p. Como na implementação alocamos o cabeçalho EP nos campos $VLAN_{VID}$ e MAC Ethernet de origem e destino, que somam 108 bits, podemos observar que o EP alocaria com folgas as configurações genéricas propostas.

Diâmetro *d*	**5**	**7**	**9**
p=16	20	28	36
p=32	25	35	45
p=64	30	42	54
p=128	35	49	63

Tabela 1: Tamanho de cabeçalhos(em *bits*) para diferentes configurações.

Topologia	**Portas *p***	***Hosts***	**Diâmetro *d***	**Diâmetro médio**
Fat-tree-3	24	3,456	6	5.9
Fat-tree-4	8	4,096	8	7.6
DCell-1	58	3,422	5	4.9
DCell-2	7	3,192	11	8.2
BCube-1	58	3,364	4	3.9
BCube-3	5	3,125	10	8.0

Tabela 2: Configurações de topologias.

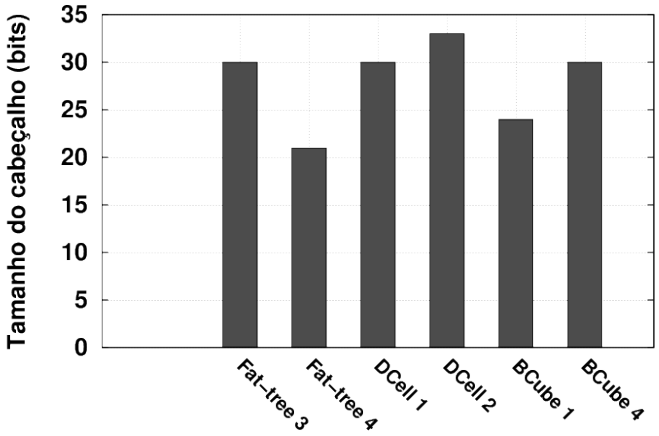


Figura 23: Tamanho do cabeçalho EP para alguma topologias utilizadas em *data center*.

Agora, avaliamos os tamanhos de cabeçalho necessários para topologias utilizadas pelas principais arquiteturas de redes de *data center* encontradas na literatura cor-

rente (COUTO; CAMPISTA; COSTA, 2012), *fat-tree*, DCell, e BCube, com variações nos parâmetros para manter o número de servidores constante e próximo a 3500. A Tabela 2 mostra os parâmetros da topologia, bem como o número correspondente de portas por *switch* e o número total de *hosts*. A tabela também inclui propriedades da rede tais como o diâmetro e o comprimento médio dos caminhos mais curtos entre os servidores.

Vale a pena notar que, embora algumas destas topologias possam operar parcialmente completas, consideramos apenas as topologias completas, ou seja, com todos as interfaces de rede de servidores e *switches* em uso. Para construir os cabeçalhos, consideramos o cenário de pior caso em termos de distância, ou seja, escolhemos nós de origem e de destino de modo a que a distância entre eles fosse o diâmetro da rede.

A Figura 23 mostra os resultados dos tamanhos de cabeçalho resultante. Como pode ser visto, o maior tamanho de cabeçalho acontece com o DCell2, que apresenta o cabeçalho de 33 bits, que é muito menor que a quantidade de bits disponível.

4.8.2 Tempo de recuperação de falhas

Primeiramente, será avaliado o mecanismo de restauração de caminho com uso de caminhos pré-computados. O objetivo é comparar a vantagem de se usar as rotas previamente calculadas no controlador em relação ao tempo de restabelecimento dos fluxos. Para isso, comparamos a abordagem com rotas pré calculadas com outra na qual o novo caminho é calculado reativamente, após a falha ser detectada.

Sobre uma topologia *fat-tree*, do mesmo tamanho da mostrada na Figura 22, um tráfego UDP foi iniciado entre dois *hosts*, escolhidos de modo que a distância entre eles fosse de 6 saltos (pior caso correspondente ao diâmetro da rede), e foram introduzidos um número variável de falhas em enlaces aleatórios. No caso em que pelo menos uma das falhas caísse no caminho selecionado entre o emissor e receptor, mediu-se o tempo necessário para restabelecer a comunicação. Note que o tempo de detecção de falha não foi considerado nestas experiências, para que a comparação fosse apenas entre os tempos de recuperação.

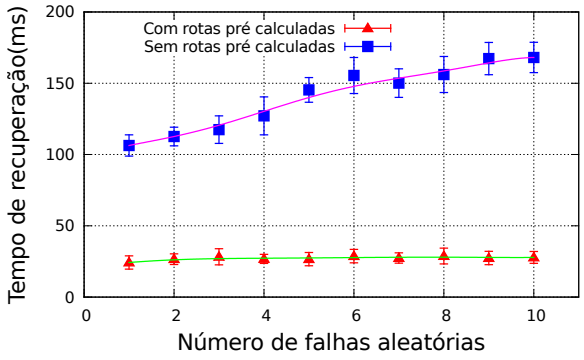


Figura 24: Tempo de recuperação com e sem caminhos pré calculados.

A Figura 24 mostra o tempo médio de recuperação em função do número de falhas aleatórias após a execução de 10 experimentos. Com a utilização de caminhos pré-calculados, o tempo foi cerca de 23 ms para uma única falha, aumentando lentamente à medida que cresce o número de falhas. Já sem rotas pré-calculadas, o tempo começa em cerca de 100ms, e sobe rapidamente quando o número de falhas simultâneas aumenta. De fato, quando ocorre mais de uma falha, o controlador pode precisar repetir o procedimento de mudança do caminho codificado no ToR. Porém, a seleção de caminhos pré-calculados é claramente mais eficiente do que rodar o algoritmo de caminho mínimo, o que explica a diferença significativa no tempo de recuperação.

Agora, testamos o tempo de recuperação de falhas em outras topologias. Além da *fat-tree*, medimos na BCube e Dcell o tempo de recuperação de uma única falhas, a fim de avaliar se existe diferença no tempo de recupreção entre as topologias. De forma semelhante ao experimento anterior, iniciamos um fluxo entre dois hosts, escolhidos de forma que a distância entre ele fosse igual ao diâmetro da rede, e inserimos uma falha no caminho.

Os resultados, exibidos na Figura 25, mostram que as topologias não tem influência significativa sobre os tempos de recuperação. Isto é explicado pelo fato da implementação do *switch* e o mecanismo de recuperação de falha serem idênticos, independentemente da topologia da rede.

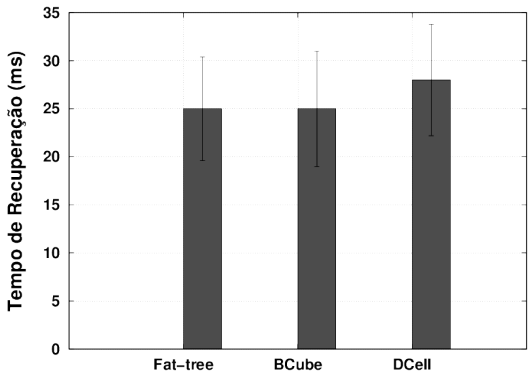


Figura 25: Comparação do tempo de recuperação em diversas topologias.

4.8.3 Desempenho de encaminhamento

Para implementar o encaminhamento, modificamos a implementação do *switch* OpenFlow para, antes de enviar um pacote pela porta de saída, fazer o deslocamento do EP, conforme descrito na Seção 4.4. Vamos agora mensurar o impacto desse processamento adicional em comparação com o desempenho de *switches* não modificados.

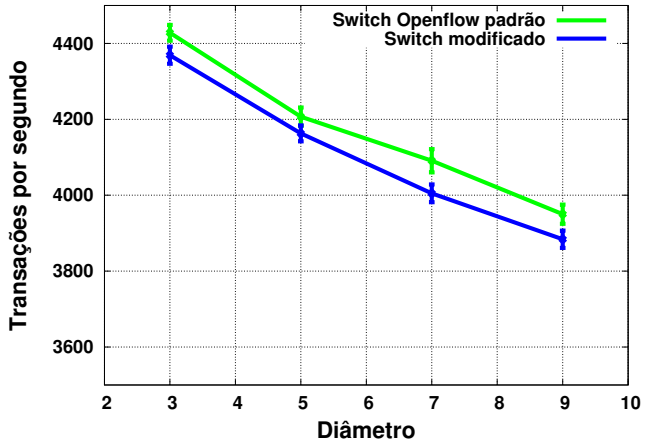


Figura 26: Desempenho do encaminhamento

Usando topologias lineares com diâmetros diferentes, usamos a ferramenta *netperf* (BIDULOCK,) para medir o número de transações *resquest/response* entre um emissor e um receptor, onde uma transação é definida como a troca de uma única requisição em nível de aplicação e uma resposta. A partir da taxa de transações, pode-se inferir a latência média de ida e volta.

Como mostrado na Figura 26, a taxa média de transações para o *switch* OpenFlow modificado é bem próxima ao do *switch* sem modificações, sendo que para os comprimentos mais curtos diferença é menor que o desvio padrão. Isso mostra que as operações de deslocamento do cabeçalho têm um baixo impacto no desempenho dos *switches*.

4.8.4 Balanceamento de carga

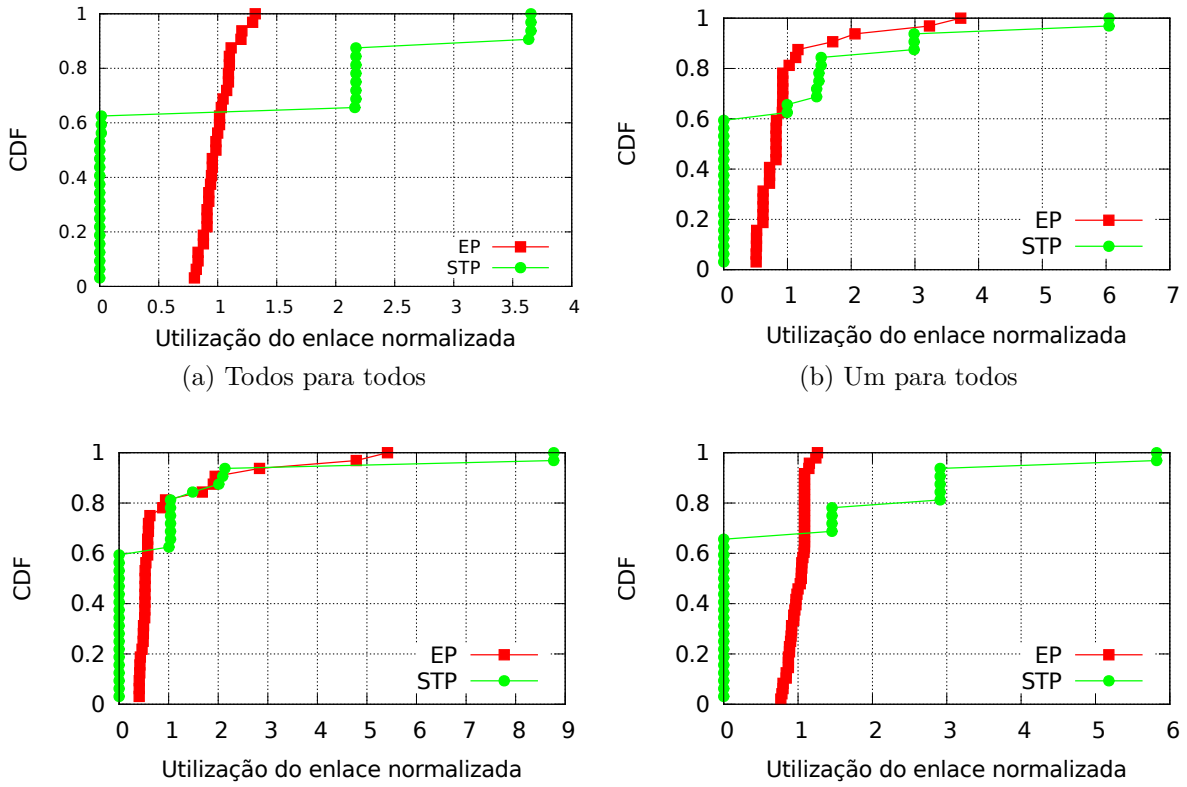
Para investigar a capacidade de balanceamento de carga do *oblivious routing* com encaminhamento EP, utilizou-se em nosso ambiente de testes diferentes matrizes de tráfego para avaliar como o tráfego é distribuído entre os links disponíveis. A distribuição de carga da implementação é comparada com o *Spanning Tree* (STP) considerando novamente um *fat-tree* (Figura 22). Quatro tipos de tráfego foram testados: (a) todos para todos, (b) um para todos, (c) todos para um, e (d) tráfego aleatório(8 *hosts* de origem, cada um enviando pacotes a outros 8 *hosts*). Usamos a ferramenta *iperf* para gerar os fluxos TCP.

Os gráficos na Figura 27 mostram a distribuição normalizada da carga nos enlaces após 10 (dez) experimentos. A carga normalizada é definida pela equação:

$$c_i = \frac{N \times X_i}{\sum_{i=1}^{N} X_i}$$

na qual X_i denota o número de pacotes que foram transportados no enlace i, e N é o número de enlaces.

Como esperado, o STP sobrecarrega alguns enlaces e deixa boa parte sem utilização. Por outro lado, a abordagem com o *oblivious routing* espalha o tráfego de modo mais uniforme através da rede. Podemos ainda observar que com STP, em todos os casos de teste, 60% dos enlaces são completamente inutilizados. Já com a implementação do *oblivious routing*, a carga teve uma ótima distribuição nas matrizes de tráfego "todos para todos"e "aleatório", com pequenas variações em torno do valor normalizado ideal, que é 1 (Figura 27(a) e Figura 27(d)). Nos casos de "um para todos"e "todos para um", apesar da tendência natural do tráfego em sobreutilizar alguns enlaces, ainda assim houve uma vantagem significativa nos enlaces mais sobrecarregados da abordagem EP comparada ao STP (Figura 27(b) e Figura 27(c)).



(a) Todos para todos

(b) Um para todos

(c) Todos para um

(d) Aleatório

Figura 27: Avaliação do Balanceamento de carga. CDF da utilização dos enlaces

4.8.5 Análise de requisitos de estado

A última comparação considera os requisitos de estado entre Portland, VL2, SiBF e EP. Como estes trabalhos destacam a implementação sobre *fat-trees*, tomamos esta topologia como exemplo, considerando uma rede com três camadas, com cada ToR ligado

Hosts físicos	2.880				23.040				103.608			
Racks	144				1152				5148			
Nível de agregação	24 (p1=24)				96 (p1=48)				144 (p1=144)			
Nível de núcleo	12 (p2=24)				24 (p2=96)				72 (p2=144)			
	VL2	Portland	SiBF	EP	VL2	Portland	SiBF	EP	VL2	Portland	SiBF	EP
Entradas no ToR	200	120	120	120	1292	120	120	120	5420	120	120	120
Entradas no agrer.	180	24	24	24	1272	48	48	48	5400	144	144	144
Entradas no núcleo	180	24	24	24	1272	96	96	96	5400	144	144	144

Tabela 3: Análise de Estado.

a 20 servidores e 2 *switches* AGGRs. As (p_1) portas dos *switches* AGGRs são usadas para conectar a $p_1/2$ ToRs e $p_1/2$ *switches* de núcleo, cada um com p_2 links.

Dependendo dos valores de p_1 e p_2, a matriz de conectividade pode variar de, por exemplo, 3.000 ($p_1 = 24$) até 100.000 ($p_1 = 144$) nós físicos. Devido à abordagem de roteamento na origem, os requisitos da tabela de fluxo são mínimos e constantes nos *switches* intermediários, ou seja, apenas uma entrada por interface, como mostra a Tabela 3 para Portland, SiBF e EP. Por outro lado, VL2 requer entradas de encaminhamento proporcionais ao número de *switches* de modo a encaminhar pacotes ao longo dos dois níveis de encapsulamento IP.

Para o EP, o aumento da escala não afeta o número de entradas de fluxo nos *switches*, que é constante e igual ao número de *switches* vizinhos. Nos ToRs, a quantidade de entradas de fluxo cresce com o número de fluxos de saída simultâneas (assumido cinco por VM) mais uma quantidade constante de entradas (1 por VM), a fim de realizar a reescrita do MAC e a entrega correta dos pacotes ao destino.

5 Roteamento na origem com caminhos alternativos

Partindo da proposta de caminho codificados apresentada no capítulo anterior, este capítulo vem propor uma extensão da ideia com o objetivo de melhorar o tempo de reação às falhas incluindo caminhos alternativos no cabeçalho do pacote. Nessa abordagem, o controlador definirá as regras no *switch* de origem para, além do caminho principal, colocar um número de caminhos alternativos no cabeçalho do pacote.

Assim, em uma situação de falhas nos enlaces, o *switch* pode procurar nos cabeçalhos por caminhos alternativos por onde o pacote pode ser enviado, tornando mais ágil a recuperação de falhas, evitando que o pacote seja enviado ao plano de controle.

Esse mecanismo, intitulado **SlickFlow**, implementado também com OpenFlow, foi inspirado do trabalho Slick Packets (NGUYEN et al., 2011). Veremos que esse mecanismo exigirá outras alterações na implementação do *switch* OpenFlow padrão, bem como a redefinição dos cabeçalhos dos caminhos codificados.

5.1 Visão Geral

A proposta de caminhos codificados na origem pode ser estendida para melhorar o tempo de reação às falhas incluindo caminhos alternativos no cabeçalho do pacote. Nessa abordagem de proteção de caminho, o controlador define a regra no *switch* de origem para, além do caminho principal, colocar um número de caminhos alternativos no cabeçalho do pacote. Assim, em uma situação de falhas nos enlaces, o *switch* pode examinar no cabeçalho EP se existe um caminho alternativo por onde o pacote pode ser enviado. Esse mecanismo torna mais ágil a recuperação de falhas, pois evita que o pacote seja enviado ao plano de controle, para que as regras sejam mudadas no *switch* de origem. Porém, como veremos, requer mais espaço no cabeçalho para abrigar os diversos caminhos.

5.1.1 Princípios de projeto

Sendo o SlickFlow derivado da proposta de encaminhamento com EP, sua arquitetura preserva as características discutidas no capítulo anterior, a lembrar:

- **Separação da computação de rotas do tratamento de falhas**

- **Controle centralizado**
- **Roteamento na origem**
- **Separação do identificador do localizador**

Para novos fluxos na borda da rede, o controlador comanda o *switch* de origem a embutir no pacote um caminho primário e caminhos alternativos em forma de um cabeçalho SlickFlow. Este mecanismo de proteção de caminho permite que o tráfego seja transferido para um caminho alternativo quando for detectada uma falha no caminho primário. Isso contrasta com mecanismos reativos de restauração caminho, em que o caminho alternativo é estabelecido pelo controlador quando este recebe uma sinalização de falha dos *switches*.

A Figura 28 mostra a visão geral do SlickFlow. De forma semelhante ao que acontece com encaminhamento com EP, quando um emissor s deseja enviar um pacote para um destino d e não existe uma regra na tabela fluxo no *switch* ToR de origem $S7$, e pacote é encaminhado ao controlador NC (passo I). O NC seleciona o caminho principal ($S7 - S4 - S2 - S6 - S10$) e caminhos alternativos ($S7 - S3 - S1 - S5 - S10$) e ($S4 - S1 - S5 - S10$), então instala uma regra nos ToR de origem e de destino (passo II). A regra no ToR de origem $S7$ (i) anexa os caminhos selecionado ao cabeçalho dos pacotes (passo III), e (ii) encaminha o pacote para S4. Nos *switches* intermediários as decisões de encaminhamento são baseadas nos caminhos contidos nos cabeçalhos (passo IV). No caso de um evento de falha, o *switch* procura por um caminho alternativo no cabeçalho SlickFlow, se disponível, o pacote é redirecionado para o caminho de *backup* sem intervenção do NC. Finalmente, a regra no ToR de destino $S10$ restaura o cabeçalho original e entrega o pacote para o destino d (passo V).

A seguir, descrevemos os detalhes dos pressupostos e mecanismos subjacentes.

5.2 Seleção de caminhos

Uma vez que o controlador precisa instalar regras para um novo fluxo, ele seleciona, entre os caminhos previamente calculados entre a origem e o destino, o caminho ao longo do qual deseja que o pacote seja encaminhado, e selecionará também caminhos alternativos disjuntos.

Os caminhos que o controlador escolherá podem ser definidos a partir de vários critérios. Podem selecionados, por exemplo, caminhos que evitem falhas únicas de enlaces ao longo de um caminho primário, que antecipem falhas de nós, otimizem a rede segundo métricas como largura de banda, etc. No trabalho, consideramos que os caminhos serão

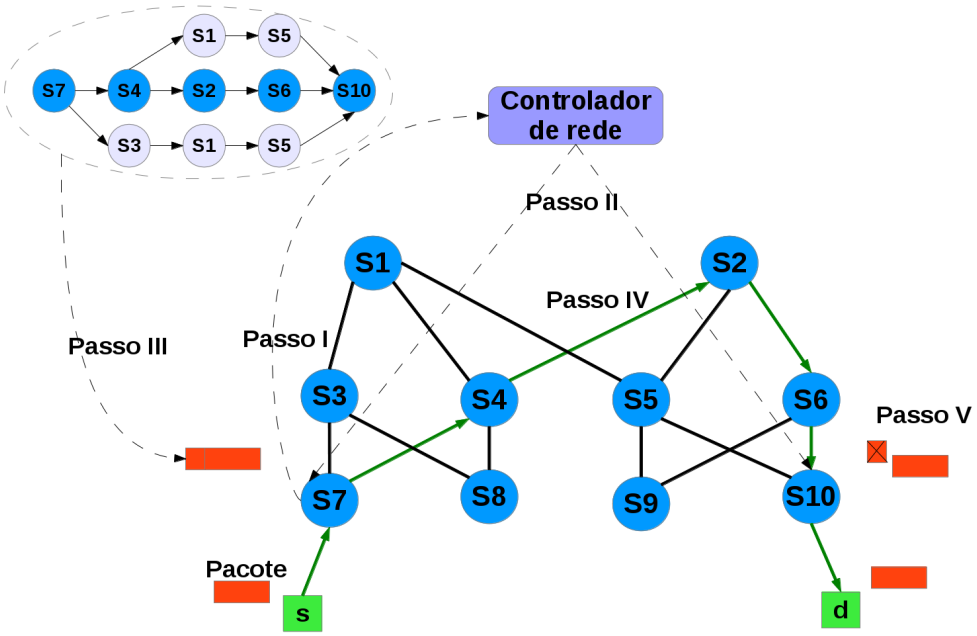


Figura 28: Visão geral do SlickFlow.

selecionados de forma a minimizar a latência no caminho primário e oferecer caminhos alternativos para evitar falhas únicas nos enlaces.

Primeiro, o NM seleciona o caminho principal P a partir de uma origem s até um destino d, então, para cada salto (ou um subconjunto do saltos) de P, é selecionado um caminho alternativo P_i, por onde o pacote será encaminhado se o próximo salto do caminho principal não estiver disponível. Um caminho alternativo P_i é um caminho que começa no nó i de P e vai até o destino d sem passar pelo enlace que liga i ao próximo nó do caminho principal. Para diminuir a latência do caminho primário, este caminho é selecionado pelo NM como o caminho mais curto entre s e d. No caso de haver múltiplos caminhos mais curtos para o destino, o controlador pode arbitrariamente selecionar um desses caminhos.

Apesar da seleção de caminho apresentada cobrir apenas falhas de um único enlace, é possível recuperar múltiplas falhas em um caminho com a ajuda do controlador centralizado. O controlador instala um caminho alternativo no ToR de depois de receber a notificação de falha dos *switches*.

5.3 Codificação

Após a seleção dos caminhos, o controlador deve codificá-los no o cabeçalho EP como uma sequência de segmentos que serão utilizados por cada *hop* ao longo do percurso.

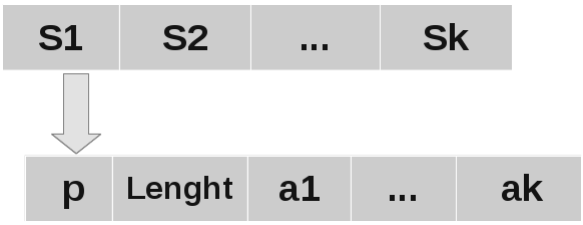


Figura 29: Codificação do EP com caminhos alternativos.

De forma semelhante ao cabeçalho descrito na Seção 4.3, o EP com caminhos alternativos é representado com uma sequencia de segmentos que serão utilizados por cada salto ao longo do caminho para realizar o encaminhamento. Porém, além do próximo salto do pacote, o segmento indica também mais duas informações. Como mostrado na Figura 29, um segmento correspondente a um nó k indica: (I) o próximo salto no caminho principal; (II) o comprimento do caminho alternativo de k e; (III) o caminho alternativo de k, representado por uma sequência de rótulos. O cabeçalho contém ainda uma informação adicional, um campo de um bit, que chamamos de *alternativo*, que especifica se o pacote está ou não em um caminho alternativo. A Figura 30 mostra o exemplo de codificação de um cabeçalho SlickFlow.

Esta codificação é baseada nos estudos mostrados em (NGUYEN et al., 2011), onde os autores compararam um formato de codificação semelhante ao apresentado acima com a outro formato que representa a rede como grafos acíclicos. A escolha baseia-se no fato de que a codificação através de grafos resulta em um cabeçalho menos compacto e aumentam a complexidade da operação de encaminhamento. Observe também que a forma de codificação pode ser generalizada para permitir que mais de um caminho alternativo para o mesmo nó, ou mesmo permitir caminhos alternativos para os caminhos alternativos. No entanto, a investigação de formatos de codificação avançadas e outras otimizações foram deixados para trabalhos futuros.

Discutiremos a seguir como os roteadores usam esta informação para encaminhar pacotes.

5.4 Encaminhamento

Ao receber um pacote, o *switch* extrai o código contido no primeiro segmento(mais a esquerda) e o usa como chave de busca na tabela de fluxos para se obter a porta de saída. Depois disso, ele checa se o enlace está disponível e verifica o bit *alternativo* para saber se ele está no caminho principal ou em um caminho alternativo.

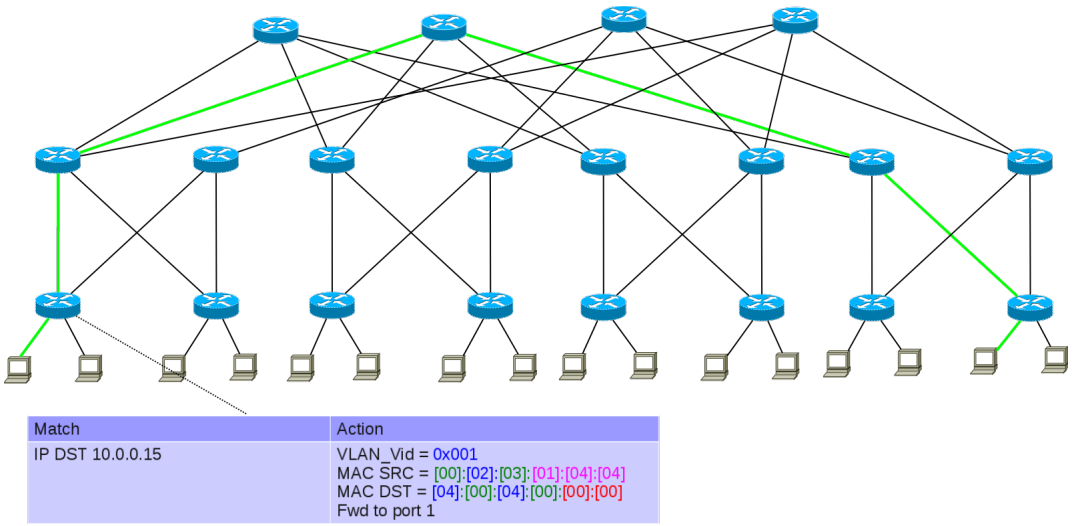


Figura 30: Exemplo do cabeçalho codificado

Switch no caminho principal

Se o *switch* estiver no caminho principal e o enlace de saída correspondente ao próximo salto estiver disponível, o *switch* apaga o primeiro segmento, deslocando o resto do cabeçalho para a esquerda, de forma que o segundo segmento passa a ser o primeiro, e assim sucessivamente. O pacote é então enviado para o próximo salto no caminho principal com o cabeçalho atualizado.

Se o enlace com o próximo salto do caminho primário não estiver disponível, o *switch* verifica o comprimento do seu caminho alternativo. Se for 0, ou seja, não existe caminho alternativo, o pacote é descartado. Caso contrário, o *switch* lê o rótulo do próximo salto do caminho alternativo a_1. Se o *link* correspondente a a_1 não está disponível, o pacote é descartado. Se o *link* está disponível, o *switch* remonta o cabeçalho, substituindo o rótulos do caminho principal pelos rótulos caminho alternativo $(a_2, ..., dl)$. Ele também muda o bit *alternativo* para o valor 1. O pacote é então enviada para o próximo salto d_1.

Switch no caminho alternativo

Se o bit *alternativo* estiver com o valor 1, o *switch* sabe que o pacote está em um caminho alternativo. O rótulo do próximo salto é lido, e se o enlace correspondente não está disponível, o pacote é descartado. Se o *link* está disponível, o *switch* desloca o cabeçalho para a esquerda e encaminha o pacote para o próximo salto.

Quando um enlace indisponível volta a ficar ativo, o fluxo é enviado novamente para o caminho principal automaticamente. Como o *switch* verifica que o próximo salto do caminho principal está disponível novamente, ele simplesmente envia o pacote para esse caminho, sem qualquer sinalização adicional.

5.5 Implementação

Implementamos o mecanismo de encaminhamento do SlickFlow com *switches* OpenFlow (Open vSwitches), e o Network Controller(NC) foi desenvolvido como uma aplicação sobre o NOX (GUDE et al., 2008).

Quando um pacote chega ao ToR e não existe uma entrada correspondente na tabela de fluxo, este é enviado para o NC, que seleciona os caminhos primários e alternativos entre todos os caminhos pré-calculados entre a origem e o destino. Os NC em seguida, instala as regras necessárias aos ToR's de origem e destino. Na ToR de origem, a regra inclui uma ação para adicionar o cabeçalho SlickFlow ao pacote, e no ToR de destino as ações de fluxo de instruem o *switch* a remover o cabeçalho SlickFlow e entregar o pacote para ao destino final.

O encaminhamento nos *switches* intermediários é baseado nas regras fixas instaladas pelo NC durante a inicialização e descoberta de topologia. Estas regras baseiam-se no rótulo de próximo salto do caminho principal e a porta de saída para a qual o pacote deve ser enviado. O *software* padrão do *switch* OpenFlow foi modificado para, antes de enviar um pacote pela porta de saída, fazer o processamento do cabeçalho SlickFlow descrito na seção anterior.

Colocamos o cabeçalho SlickFlow nos campos MAC Ethernet e VLAN. O rótulo do próximo salto do primeiro segmento é colocado no campo $VLAN_{VID}$ e o restante do cabeçalho é colocado nos 96 bits dos campos MAC Ethernet de origem e destino, e o bit *alternativo* colocado no campo de $VLAN_{PCP}$. Para esta configuração, o comprimento de cada rótulo tem um tamanho fixo de 4 bits, de modo que cada *switch* pode ter no máximo 16 vizinhos e os caminhos alternativos podem ter no máximo 16 saltos.

Reconhecemos que esta configuração é insuficiente para cenários de *data centers* reais que podem conter *switches* com mais de 100 portas. No entanto, é suficiente como prova de conceitos, para demonstrar a ideia e provar a viabilidade da implementação. Um tamanho de rótulo mais adequado teria 8 bits de comprimento, permitindo que cada possuísse *switch* até 256 vizinhos, porém, na implementação, os bits do espaço de endereços MAC Ethernet se esgotariam rapidamente, não permitindo codificar caminhos alternativos (ou limitando a caminhos muito curtos). Um cenário que estamos considerando para trabalhos futuros, aproveitando as últimas versões do protocolo OpenFlow, é usar espaço de bits adicionais nos cabeçalhos dos pacotes (por exemplo, PBB (LEE; KIM; SONG, 2010), MPLS (SHARMA; HELLSTRAND, 2003) e/ou IPv6 básico e cabeçalhos de extensão) para permitir cabeçalhos SlickFlow maiores.

Uma possibilidade óbvia de reduzir o tamanho do cabeçalho é não incluir o caminho alternativo do *switch* de origem, onde regras de codificação SlickFlow estão instalados. Em vez disso, o caminho de alternativa em relação ao primeiro salto pode ser estabelecido

pelo NC como uma entrada de fluxo adicional com prioridade mais baixa. Quando uma falha em um link de saída do primeiro *switch* é detectado, este pode desabilitar a porta de saída afetada e desviar o fluxo automaticamente para o caminho alternativo. No caso de *switches* com suporte ao OpenFlow 1.1+, tabelas de grupos com ações *fast-failover* podem ser utilizadas (Consortium, OpenFlow, 2011).

5.6 Avaliação

Avaliamos o SlickFlow usando o protótipo baseado em OpenFlow usando o Mininet (LANTZ; HELLER; MCKEOWN, 2010). Os principais objetivos de avaliação incluem explorar: (I) como topologias arbitrárias podem ser suportadas e como se comportam no SlickFlow, e (II) o quão rápido o plano de dados pode se recuperar de falhas de links na rede. Consideramos topologias comuns em DCN's variando sua configuração e analisando quantos bits seriam necessários para suportar a codificação proposta.

Como prova do conceito, SlickFlow é implementado e avaliado codificando caminhos alternativos para cobrir falhas de um único enlace caminho primário (Secção 5.2), medindo o tempo de recuperação de falhas, perda de pacotes, e desempenho de encaminhamento.

Não avaliamos o balanceamento de carga e requisitos de estados pois não existem diferenças em relação ao apresentado no capítulo 4.

5.6.1 Tamanho do cabeçalho codificado

Começamos avaliando o tamanho do cabeçalho SlickFlow usando o formato de codificação apresentado na Seção 5.3. O tamanho do cabeçalho depende principalmente do diâmetro da topologia e do número de portas de saída dos *switches* Assim, considerando uma rede com diâmetro d, cada *switch* com p portas e cada caminho alternativo com comprimento de l bits, o cabeçalho SlickFlow terá d segmento, cada um com o tamanho de $\log_2(p) + l$, mais o tamanho do caminho alternativo. Para simplificar a equação, assumimos que para cada caminho entre os nós existe um caminho alternativo de mesmo comprimento. Assim, o caminho alternativo para o primeiro salto tem comprimento igual ao diâmetro da rede d, o caminho alternativo relativo ao próximo salto contém $d - 1$ rótulos, e assim por diante. O nível de redundância n é definido com o número de caminho alternativos para cada segmento. Essas suposições levam à seguinte equação para o tamanho do cabeçalho SlickFlow, em bits:

$$S = d * (\log_2(p) + l) + n * \sum_{i=1}^{d} i * \log_2(p) \tag{5.1}$$

Nível de redundância *n*	**1** (caminho alternativo)			**2** (caminhos alternativos)		
Diâmetro *d*	**5**	**7**	**9**	**5**	**7**	**9**
p=16	80	140	216	120	224	360
p=32	95	168	261	145	273	441
p=64	110	196	306	170	322	522
p=128	125	224	351	195	371	603

Tabela 4: Tamanho de cabeçalhos(em *bits*) para diferentes configurações.

Topologia	**Portas *p***	***Hosts***	**Diâmetro *d***	**Diâmetro médio**
Fat-tree-3	24	3,456	6	5.9
Fat-tree-4	8	4,096	8	7.6
DCell-1	58	3,422	5	4.9
DCell-2	7	3,192	11	8.2
BCube-1	58	3,364	4	3.9
BCube-4	5	3,125	10	8.0

Tabela 5: Configurações de topologias.

$$S = d * (\log_2(p) + l) + n * \frac{d^2 + d}{2} * \log_2(p) \tag{5.2}$$

A primeira componente da equação se refere à estrutura do segmento (p e o comprimento l na Figura 29) e a segunda componente se refere à redundância contida no segmento (a_i na Figura 29). Assumindo que o rótulo correspondente ao comprimento do caminho alternativo tem 4 bits (que significa que os caminhos alternativos terão no máximo 16 saltos), a Tabela 4 mostra o tamanho em bits do cabeçalho SlickFlow para algumas topologias, variando o diâmetro d, número de portas por swtich p, e quantidade da caminho alternativos por salto n (nível de redundância).

Como podemos ver, o diâmetro da topologia é o fator dominante, como era esperado ao se analizar a equação 5.2. Podemos observar também que com nível de redundância n igual a 2, ou seja, dois caminhos alternativos por segmento, o tamanho do cabeçalho cresce rapidamente. Assim, para fins de validação, assumimos o nível de redundância igual a 1.

Agora, avaliamos os tamanhos de cabeçalho necessários para topologias *fat-tree*, DCell, e BCube. Mostramos novamente a Tabela 5, que mostra os parâmetros das topologias, bem como o número correspondente de portas por *switch* e o número total de *hosts*, mantendo o número de servidores constante e próximo a 3500. Lembrando que assumimos topologias completas.

A Figura 31 mostra os resultados dos tamanhos de cabeçalho resultante. Como pode ser visto, BCube1 tem o menor tamanho, resultando em uma codificação de 84 bits, enquanto que na DCell2 o cabeçalho é de cerca de três vezes maior.

Um resultado interessante é observado comparando topologias do mesmo tipo com parâmetros diferentes, tais como BCube1 com BCube4, e DCell1 com DCell2. Podemos ver que topologias com maior número de portas por *switch* têm tamanhos de cabeçalho

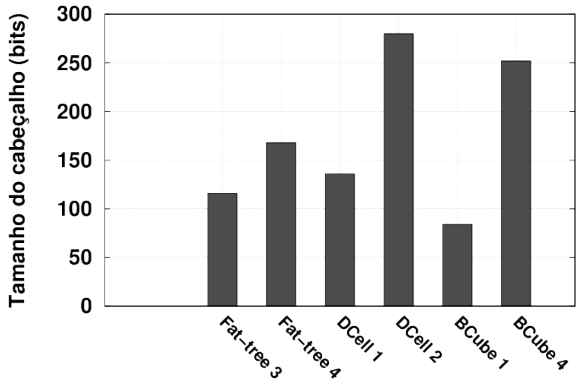


Figura 31: Tamanho do cabeçalho SlickFlow.

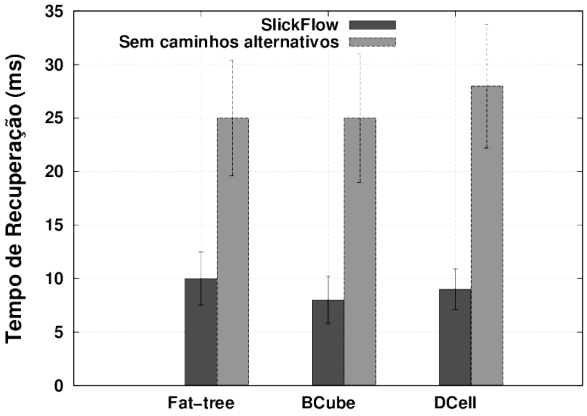


Figura 32: Tempo de recuperação.

maiores. A razão é que os pares origem-destino têm caminhos primários maiores, exigindo caminhos alternativos para um maior número de nós. Por outro lado, estas topologias são mais resistentes, pois oferecem um maior número de caminhos entre cada par de nós.

5.6.2 Tempo de recuperação de falhas

A fim de avaliar o desempenho da recuperação de falhas do SlickFlow, o comparamos com o método de restauração de caminhos apresentado no capítulo anterior, que utiliza o controlador como o elemento central para tomar a decisão de restaurar o caminho do fluxo de pacotes, ou seja, o caminho alternativo é instalado nos *switches* ToR pelo controlador quando este recebe a notificação de falhas de enlace pelos *switches* OpenFlow (evento *port-down*).

No primeiro experimento, medimos o tempo para restabelecer o tráfego para uma

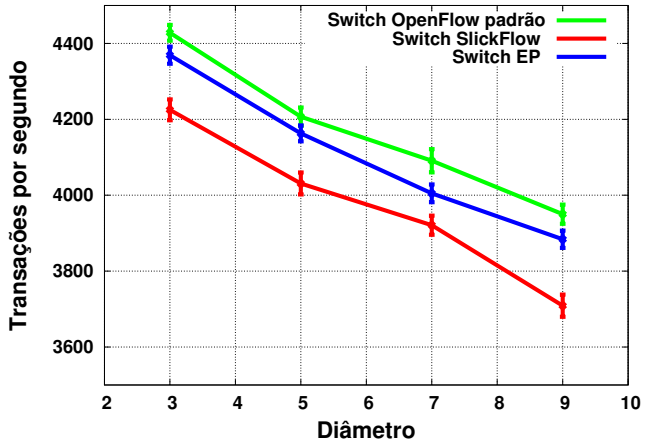


Figura 33: Desempenho do encaminhamento.

única falha de link. Um tráfego UDP de 10 Mbps é iniciada entre os dois *hosts* arbitrários. Monitoramos a atividade do receptor, enquanto que injetamos uma falha no caminho e medimos o tempo necessário para restabelecer a comunicação.

A figura Figura 32 mostra o tempo médio de recuperação e desvio padrão para as topologias após 10 experimentos. O tempo de recuperação do SlickFlow é aproximadamente 10 ms, enquanto que o tempo para o processo de restauração de caminhos é cerca de 150% maior. A diminuição do tempo de recuperação se deve ao fato de que o mecanismo de recuperação depende do RTT entre os *switches* e o controlador. É importante lembrar ainda que para redes maiores, com tráfego de controle mais intenso, o RTT efetivo tende a ser maior, diminuindo o desempenho do mecanismo de restauração (CURTIS et al., 2011).

Idealmente, o SlickFlow deveria ter um tempo de recuperação próximo de zero, sem nenhuma perda de pacotes, uma vez que cada pacote é processado pelos *switches*, e se o *link* do caminho primário não estiver disponível, o pacote deve ser enviado para o caminho alternativo. Em nossa implementação da prova de conceito baseada em *software*, no entanto, a perda nula de pacotes ocorre apenas para as taxas de transmissão mais baixas (1Mbps). Este comportamento de perda de pacotes pode ser explicado devido ao tempo que leva para um *switch* detectar mudanças de *status* da porta e atualizar as *flags* que são usadas para verificar a atividade da porta de saída.

5.6.3 Desempenho de encaminhamento

Da mesma forma que o encaminhamento EP, mostrado no capítulo anterior, o SlickFlow também requer que o *switch* padrão OpenFlow seja modificado. Além do deslocamento do cabeçalho a cada salto, antes de enviar o pacote pela porta de saída, o

comutador SlickFlow deve processar o cabeçalho do pacote para verificar se o enlace de saída está disponível e saber se o pacote está no caminho principal ou alternativo, como descrito na Seção 5.4.

Partindo de topologias lineares, foram medidos o número de transações *resquest/response* entre um emissor e um receptor. Como mostrado na Figura 33, comparando com o desempenho da implementação do capítulo anterior o SlickFlow tem uma queda de rendimento de aproximadamente 5%. Isso acontece pois no SlickFlow o *switch* precisa fazer verificações que no encaminhamento com EP sem caminhos alternativos não eram necessárias. É preciso verificar o bit *alternativo* para saber se o pacote está no caminho principal ou em um caminho alternativo, além de verificar se o enlace de saída está disponível.

Vale ressaltar que os experimentos foram realizados utilizando um protótipo implementado em *software*, e que não possuem um nível de precisão de uma implementação em *hardware*. Todavia, os valores apresentados são válidos para efeito de comparação entre as abordagens, pois permitem analisar a ordem de grandeza e a tendência de crescimento dos resultados apresentados.

6 Conclusão e Trabalhos Futuros

6.1 Contribuições

Neste trabalho, é apresentada uma abordagem de roteamento na origem para redes de *data center* para alcançar escalabilidade e tolerância a falhas com base no OpenFlow. A proposta funciona através da codificação de caminhos inseridos no cabeçalho dos pacotes. Foi proposto também a utilização de caminhos alternativos nos cabeçalhos dos pacotes a fim de diminuir o tempo de recuperação de falhas.

Os resultados obtidos através da avaliação conduzida, baseados em experimentos e análises no ambiente de prototipação, demonstram o potencial das ideias desenvolvidas no contexto de redes de *data center*. Alguns pontos podem ser destacados:

- Escalabilidade: Para alcançar a escala da ordem de milhares de servidores, dois aspectos precisam ser considerados: tamanho das tabelas de encaminhamento e o volume de tráfego de *broadcast*. Empregando o controle centralizado e o roteamento na origem alcançamos a escalabiblidade do plano de dados, uma vez que os comutadores requerem apenas o conhecimento local dos seus vizinhos, levando a tabelas com tamanho proporcional ao número de vizinhos. Além do mais, utilizamos o controlador para reduzir as operações de *broadcast* e *flooding*.

- Recuperação de falhas: Este trabalho traz a sua principal contribuição no que diz respeito à recuperação de falhas, pois propõe, implementa e avalia dois mecanismos de recuperação de falhas que dispensam o uso de algoritmos distribuídos: (i) empregando o controlador centralizado para mudar os caminhos codificados nos *switches* de borda, e (ii) utilizando caminhos alternativos no cabeçalho dos pacotes para que a recuperação ocorra sem a intervenção do controlador. No primeiro caso, mostramos que a utilização dos caminhos pré-calculados é importante para reduzir o tempo de recuperação. Em seguida, mostramos que é possível reduzir ainda mais esse tempo com caminhos alternativos nos cabeçalhos, evitando o tempo necessário para o controlador receber a informação de falhas e trocar as regras de codificação de caminhos nas bordas. Com o SlickFlow houve uma redução de até 60% no tempo de recuperação de falhas.

A utilização de caminhos alternativos nos cabeçalhos dos pacotes resulta em um aumento na quantidade de bits necessária para realizar a codificação. Assim, existe um *trade-off* entre o nível de redundância oferecido e o tamanho dos cabeçalhos. Equações analíticas foram obtidas para estabelecer a relação de crescimento entre o tamanho da codificação e o diâmetro da rede, considerando topologias distintas e diferentes níveis de redundância.

Na implementação, utiliza-se um espaço para o cabeçalho de tamanho limitado, o que impõe barreiras à utilização da proposta em termos de tamanho de topologias e nível de redundânca que pode ser oferecido. Para superar essa limitação, uma possibilidade é a utilização dos campos do IPv6.

- Facilidade de configuração: O comportamento de modo *plug-and-play* do protocolo Ethernet foi conservado, pois o controlador aprende a localização dos *hosts* finais, levando a baixa sobrecarga de configuração. Além disso, a utilização de endereços planos permanentes simplificam o tratamento da mobilidade e políticas de controle de acesso. A abordagem proposta leva a uma modificação na semântica dos endereços, separando o identificador do localizador. O endereço IP é usado apenas como identificador, enquanto que o localizador é o próprio caminho codificado no cabeçalho do pacote.

  Ainda, nosso mecanismo de encaminhamento não necessita de modificações nos *hosts* finais, ou necessidade de encapsulamento. Logo, sistemas operacionais e aplicativos são suportados sem necessidade de modificação em nenhuma camada.

- Balanceamento de carga: Diferentemente do Ethernet, em que todo o tráfego atravessa uma única *spanning tree*, tornando o encaminhamento mais sujeito a falhas e levando a enlaces com cargas irregulares, a abordagem proposta permite a utilização dos múltiplos caminhos disponíveis na rede. Empregamos ainda o balanceamento de carga usando *oblivious routing*, ou seja, o roteamento de pacotes de forma aleatória independentemente do tráfego. Vimos que essa abordagem simples funciona bem para espalhar o tráfego pelos diversos caminhos disponíveis.

No decorrer da pesquisa que culminou na presentes dissertação, foram elaborados dois artigos relacionados ao tema. (RAMOS; MARTINELLO; ROTHENBERG, 2013a) já publicado, e (RAMOS; MARTINELLO; ROTHENBERG, 2013b) aceito para publicação.

6.2 Trabalhos Futuros

Como trabalhos futuros, pretendemos melhorar a implementação nos seguintes pontos:

- O protótipo implementado em *software* têm o propósito de verificar que o conceito proposto é viável. Assim, sabemos que números obtidos são indicadores e uma avaliação em equipamentos com suporte a OpenFlow precisa ser efetuada para uma maior precisão. Não há barreira conceitual para que as propostas de encaminhamento e recuperação de falhas sejam implementadas em *hardware*, dado que envolvem operações de reescrita dos cabeçalhos simples e de tamanho fixo.

- Explorar as funcionalidades das novas versões do protocolo OpenFlow, como o mecanismo de *fast-failover* e a utilização do cabeçalho IPv6 para validar o SlickFlow em maior escala.

- Considerar o *hypervisor vSwitch* como o primeiro nó da rede, onde acontece o *match* das regras e a reescrita do cabeçalho.

- Tendo em vista que as falhas em centros de dados ocorrem com mais freqüência nos enlaces entre os *switches* de agregação e ToRs (GILL; JAIN; NAGAPPAN, 2011), em futuras revisões do projeto, pretendemos ampliar a codificação para aumentar o nível de redundância permitindo mais de um caminho alternativo nesses *links* críticos.

- Finalmente, otimizações no desempenho das operações de encaminhamento podem ser exploradas, reduzindo o tempo de busca nas tabelas de fluxo(OLIVEIRA et al., 2013), já que o cabeçalho SlickFlow já possui as sequências de portas de saída do *switches* ao longo do caminho do pacote.

Referências

AL-FARES, M.; LOUKISSAS, A.; VAHDAT, A. A scalable, commodity data center network architecture. In: *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*. New York, NY, USA: ACM, 2008. (SIGCOMM '08), p. 63–74. ISBN 978-1-60558-175-0. Disponível em: <http://doi.acm.org/10.1145/1402958.1402967>. 14, 24, 25, 46

Amazon. *Amazon EC2*. 2013. Disponível em: <http://aws.amazon.com/ec2/>. 17

ARMBRUST, M. et al. A view of cloud computing. *Commun. ACM*, 2010. ACM, New York, NY, USA, v. 53, n. 4, p. 50–58, abr. 2010. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1721654.1721672>. 23

BARHAM, P. et al. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 2003. ACM, New York, NY, USA, v. 37, n. 5, p. 164–177, out. 2003. ISSN 0163-5980. Disponível em: <http://doi.acm.org/10.1145/1165389.945462>. 23

BIDULOCK, B. *NETPERF Utility Installation and Reference Manual Version 2.3 Edition 7 Updated 2008-10-31 Package netperf-2.3.7*. 70

BODEN, N. J. et al. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 1995. IEEE Computer Society Press, Los Alamitos, CA, USA, v. 15, n. 1, p. 29–36, fev. 1995. ISSN 0272-1732. Disponível em: <http://dx.doi.org/10.1109/40.342015>. 23

CAESAR, M. et al. Dynamic route recomputation considered harmful. *SIGCOMM Comput. Commun. Rev.*, 2010. ACM, New York, NY, USA, v. 40, n. 2, p. 66–71, abr. 2010. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1764873.1764885>. 58

CARR, D. F. 2006. 2006. 17

CASADO, M. et al. Ethane: taking control of the enterprise. In: *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2007. (SIGCOMM '07), p. 1–12. ISBN 978-1-59593-713-1. Disponível em: <http://doi.acm.org/10.1145/1282380.1282382>. 30

CASADO, M. et al. Virtualizing the network forwarding plane. In: *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*. New York, NY, USA: ACM, 2010. (PRESTO '10), p. 8:1–8:6. ISBN 978-1-4503-0467-2. Disponível em: <http://doi.acm.org/10.1145/1921151.1921162>. 37

CHANG, F. et al. Bigtable: a distributed storage system for structured data. In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*. Berkeley, CA, USA: USENIX Association, 2006. (OSDI '06), p. 15–15. Disponível em: <http://dl.acm.org/citation.cfm?id=1267308.1267323>. 29

CHEVERESAN, R. et al. Characteristics of workloads used in high performance and technical computing. In: *Proceedings of the 21st annual international conference on Supercomputing*. New York, NY, USA: ACM, 2007. (ICS '07), p. 73–82. ISBN 978-1-59593-768-1. Disponível em: <http://doi.acm.org/10.1145/1274971.1274984>. 29

CISCO. *Data Center: Load Balancing Data Center Services SRND*. San Jose, CA: Cisco Systems, Inc, 2004. 17

CISCO. *Cisco Data Center Infrastructure 2.5 Design Guide*. San Jose, CA: Cisco Systems, Inc, 2007. 17

CLOS, C. A study of nonblocldng switching networks. *Belt Syst. Tech.*, 1953. v. 32, p. 406–424, 1953. 46

CONSORTIUM, O. OpenFlow 1.1 switch specification. *Can be accessed at http://openflowswitch. org/*, 2011. p. 1–56, 2011. Disponível em: <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:OpenFlow+Switch+Specification#3 14, 33

Consortium, OpenFlow. OpenFlow 1.1 switch specification. *Can be accessed at http://openflowswitch. org/*, 2011. p. 1–56, 2011. 79

COUTO, R. S.; CAMPISTA, M. E. M.; COSTA, L. H. Uma avaliação da robustez intra data centers baseada na topologia da rede. *XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2012*, 2012. maio 2012. 57, 68

CPQD. *OpenFlow 1.3 Tutorial*. 2013. Último acesso: 06/06/2013. Disponível em: <https://github.com/CPqD/ofsoftswitch13/wiki/OpenFlow-1.3-Tutorial>. 38

CURTIS, A. R. et al. Devoflow: scaling flow management for high-performance networks. *SIGCOMM Comput. Commun. Rev.*, 2011. ACM, New York, NY, USA, v. 41, n. 4, p. 254–265, ago. 2011. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/2043164.2018466>. 39, 82

DEAN, J.; GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 2008. ACM, New York, NY, USA, v. 51, n. 1, p. 107–113, jan. 2008. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1327452.1327492>. 17, 29

DECANDIA, G. et al. Dynamo: amazon's highly available key-value store. In: *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*. New York, NY, USA: ACM, 2007. (SOSP '07), p. 205–220. ISBN 978-1-59593-591-5. Disponível em: <http://doi.acm.org/10.1145/1294261.1294281>. 29

Dell. Whitepaper, *Shared Infrastructure: Scale-Out Advantages and Effects on TCO*. 2010. 23

FOSTER, N. et al. Frenetic: a network programming language. In: *Proceedings of the 16th ACM SIGPLAN international conference on Functional programming*. New York, NY, USA: ACM, 2011. (ICFP '11), p. 279–291. ISBN 978-1-4503-0865-6. Disponível em: <http://doi.acm.org/10.1145/2034773.2034812>. 38

FU, M.; LE, Z.; ZHU, Z. Bfd-based failure detection and localization in ip over obs/wdm multilayer network. *Int. J. Commun. Syst.*, 2012. John Wiley and Sons Ltd., Chichester, UK, v. 25, n. 3, p. 277–293, mar. 2012. ISSN 1074-5351. Disponível em: <http://dx.doi.org/10.1002/dac.1236>. 65

GHEMAWAT, S.; GOBIOFF, H.; LEUNG, S.-T. The google file system. In: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2003. (SOSP '03), p. 29–43. ISBN 1-58113-757-5. Disponível em: <http://doi.acm.org/10.1145/945445.945450>. 29

GILL, P.; JAIN, N.; NAGAPPAN, N. Understanding network failures in data centers: measurement, analysis, and implications. *SIGCOMM Comput. Commun. Rev.*, 2011. ACM, New York, NY, USA, v. 41, n. 4, p. 350–361, ago. 2011. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/2043164.2018477>. 86

GREENBERG, A. et al. Vl2: a scalable and flexible data center network. *SIGCOMM Comput. Commun. Rev.*, 2009. ACM, New York, NY, USA, v. 39, n. 4, p. 51–62, ago. 2009. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1594977.1592576>. 14, 28, 49, 50, 51, 59, 63

GREENBERG, A. et al. A clean slate 4d approach to network control and management. *SIGCOMM Comput. Commun. Rev.*, 2005. ACM, New York, NY, USA, v. 35, n. 5, p. 41–54, out. 2005. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1096536.1096541>. 30

GREENBERG, A. et al. Towards a next generation data center architecture: scalability and commoditization. In: *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*. New York, NY, USA: ACM, 2008. (PRESTO '08), p. 57–62. ISBN 978-1-60558-181-1. Disponível em: <http://doi.acm.org/10.1145/1397718.1397732>. 59

GUDE, N. et al. Nox: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 2008. ACM, New York, NY, USA, v. 38, n. 3, p. 105–110, jul. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1384609.1384625>. 18, 38, 62, 78

GUEDES, D. et al. Redes definidas por software: uma abordagem sistemica para o desenvolvimento de pesquisas em redes de computadores. *SBRC 2012 - Minicursos*, 2012. Ouro Preto, Brazil, maio 2012. 14, 34

GUO, C. et al. Bcube: a high performance, server-centric network architecture for modular data centers. *SIGCOMM Comput. Commun. Rev.*, 2009. ACM, New York, NY, USA, v. 39, n. 4, p. 63–74, ago. 2009. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1594977.1592577>. 14, 47, 48, 60

GUO, C. et al. Secondnet: a data center network virtualization architecture with bandwidth guarantees. In: *Proceedings of the 6th International COnference*. New York, NY, USA: ACM, 2010. (Co-NEXT '10), p. 15:1–15:12. ISBN 978-1-4503-0448-1. Disponível em: <http://doi.acm.org/10.1145/1921168.1921188>. 61

GUO, C. et al. Dcell: a scalable and fault-tolerant network structure for data centers. In: *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*. New York,

NY, USA: ACM, 2008. (SIGCOMM '08), p. 75–86. ISBN 978-1-60558-175-0. Disponível em: <http://doi.acm.org/10.1145/1402958.1402968>. 14, 48, 49, 60

HAMADA, T.; NAKASATO, N. Infiniband trade association, infiniband architecture specification, volume 1. In: *in International Conference on Field Programmable Logic and Applications*. [S.l.: s.n.], 2005. p. 366–373. 17, 23, 46, 47

HAMILTON, J. *Networking: The last bastion of mainframe computing*. 2011. Último acesso: 06/06/2013. 30

ITU-T. *ITU-T Recommendation E.800 :Definitions of terms related to quality of service*. [S.l.], 2008. 40

JOHNSON, D. B.; MALTZ, D. A. Dynamic source routing in ad hoc wireless networks. In: *Mobile Computing*. [S.l.]: Kluwer Academic Publishers, 1996. p. 153–181. 52

Juniper Networks. Whitepaper, *Network Operating System evolution*. 2010. 31

KIM, C.; CAESAR, M.; REXFORD, J. Floodless in seattle: a scalable ethernet architecture for large enterprises. *SIGCOMM Comput. Commun. Rev.*, 2008. ACM, New York, NY, USA, v. 38, n. 4, p. 3–14, ago. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1402946.1402961>. 18, 24, 26

KIM, C.; REXFORD, J. Revisiting ethernet: Plug-and-play made scalable and efficient. In: *Local Metropolitan Area Networks, 2007. LANMAN 2007. 15th IEEE Workshop on*. [S.l.: s.n.], 2007. p. 163–169. 25, 27

KOMPELLA, K. *New Take on SDN: Does MPLS Make Sense in Cloud Data Centers?* 2012. Último acesso: 06/08/2013. Disponível em: <http://www.sdncentral.com/use-cases/does-mpls-make-sense-in-cloud-data-centers/2012/12/>. 27

KOPONEN, T. et al. Onix: a distributed control platform for large-scale production networks. In: *Proceedings of the 9th USENIX conference on Operating systems design and implementation*. Berkeley, CA, USA: USENIX Association, 2010. (OSDI'10), p. 1–6. Disponível em: <http://dl.acm.org/citation.cfm?id=1924943.1924968>. 37, 38, 65

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. New York, NY, USA: ACM, 2010. (Hotnets-IX), p. 19:1–19:6. ISBN 978-1-4503-0409-2. Disponível em: <http://doi.acm.org/10.1145/1868447.1868466>. 38, 66, 79

LEE, W.; KIM, D.; SONG, H. Y. Autonomous client discovery in backbone edge bridges for multipoint pbb-te networks. In: *Proceedings of the 12th international conference on Advanced communication technology*. Piscataway, NJ, USA: IEEE Press, 2010. (ICACT'10), p. 712–716. ISBN 978-1-4244-5427-3. Disponível em: <http://dl.acm.org/citation.cfm?id=1831508.1831660>. 78

LEISERSON, C. E. Fat-trees: universal networks for hardware-efficient super-computing. *IEEE Trans. Comput.*, 1985. IEEE Computer Society, Washington, DC, USA, v. 34, n. 10, p. 892–901, out. 1985. ISSN 0018-9340. Disponível em: <http://dl.acm.org/citation.cfm?id=4492.4495>. 46, 47

LOUKISSAS, A. *Implementation and simulation of the two-level lookup*. Dissertação (Mestrado) — UC San Diego, 2008. 23

MACAPUNA, C. A. B. et al. Controle distribuído em redes de data center baseado em gerenciadores de rack e serviços de diretório/topologia. *WCGA 2010*, 2010. 2010. 54, 66

MAHAPATRA, S.; YUAN, X.; NIENABER, W. Limited multi-path routing on extended generalized fat-trees. *IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, 2012. IEEE Computer Society, p. 938–945, maio 2012. 61

MANNIE, E.; PAPADIMITRIOU, D. *Recovery (Protection and Restoration) Terminology for Generalized Multi-Protocol Label Switching (GMPLS)*. IETF, March 2006. RFC 4427 (Informational). (Request for Comments, 4427). Disponível em: <http://www.ietf.org/rfc/rfc4427.txt>. 44

MCKEOWN, N. et al. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 2008. ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>. 18, 32, 36, 62

MICROSOFT. *Windows Azure platform case studies*. 2013. Último acesso: 06/06/2013. Disponível em: <http://www.microsoft.com/windowsazure/evidence/>. 22

MILLER, R. *Tech Titans Back OpenFlow Networking Standard*. 2011. Último acesso: 06/07/2013. Disponível em: <http://www.datacenterknowledge.com/archives/2011/03/22/tech-titans-back-openflow-networking-standard/>. 19

MOY, J. *OSPF Version 2*. IETF, abr. 1998. RFC 2328 (Standard). (Request for Comments, 2328). Updated by RFCs 5709, 6549. Disponível em: <http://www.ietf.org/rfc/rfc2328.txt>. 26

MUDIGONDA, J. et al. Netlord: a scalable multi-tenant network architecture for virtualized datacenters. In: *Proceedings of the ACM SIGCOMM 2011 conference*. New York, NY, USA: ACM, 2011. (SIGCOMM '11), p. 62–73. ISBN 978-1-4503-0797-0. Disponível em: <http://doi.acm.org/10.1145/2018436.2018444>. 22

MYERS, A.; NG, T. E.; ZHANG, H. *Rethinking the Service Model: Scaling Ethernet to a Million Nodes*. 2004. 25

MYSORE, R. N. et al. Portland: a scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.*, 2009. ACM, New York, NY, USA, v. 39, n. 4, p. 39–50, ago. 2009. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1594977.1592575>. 14, 51, 53

NGUYEN, G. T. et al. Slick packets. In: *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM, 2011. (SIGMETRICS '11), p. 245–256. ISBN 978-1-4503-0814-4. Disponível em: <http://doi.acm.org/10.1145/1993744.1993769>. 14, 56, 57, 73, 76

NSF. *NSF Awards Millions to Fourteen Universities for Cloud Computing Research*. 2013. Último acesso: 06/06/2013. Disponível em: <http://www.nsf.gov/news/>. 22

OLIVEIRA, R. E. Z. de et al. Keyflow: Comutação por chaves locais de fluxos roteados na borda via identificadores globais. *XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2013*, 2013. maio 2013. 86

Open Networking Foundation. Whitepaper, *Software-Defined Networking:The New Norm for Networks*. 04 2012. Disponível em: <https://www.opennetworking.org/images/stories/downloads/white-papers/wp-sdn-newnorm.pdf>. 30, 32

PFAFF, B. et al. e.a.: Extending networking into the virtualization layer. In: *In: 8th ACM Workshop on Hot Topics inNetworks (HotNets-VIII).New YorkCity,NY(October 2009*. [S.l.: s.n.], 2009. 38

RAMOS, R. M.; MARTINELLO, M.; ROTHENBERG, C. E. Data center fault-tolerant routing and forwarding: An approach based on encoded paths. *Sixth Latin-American Symposium on Dependable Computing - LADC'2013*, 2013. 2013. 85

RAMOS, R. M.; MARTINELLO, M.; ROTHENBERG, C. E. Slickflow: Resilient source routing in data center networks unlocked by openflow. *The 38th IEEE Conference on Local Computer Networks - LNC'2013*, 2013. 2013. 85

ROTHENBERG, C. E. et al. Data center networking with in-packet bloom filters. *XXVIII Simposio Brasileiro de Redes de Computadores e Sistemas Distribuidos*, 2010. 2010. 14, 53, 54, 55

ROTHENBERG, C. E. et al. OpenFlow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. *Cadernos CPqD Tecnologia Volume 1*, 2011. p. 65–76, julho 2011. 31

SHARMA, S. et al. Viking: a multi-spanning-tree ethernet architecture for metropolitan area and cluster networks. In: *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*. [S.l.: s.n.], 2004. v. 4, p. 2283–2294 vol.4. ISSN 0743-166X. 27

SHARMA, V.; HELLSTRAND, F. *Framework for Multi-Protocol Label Switching (MPLS)-based Recovery*. IETF, fev. 2003. RFC 3469 (Informational). (Request for Comments, 3469). Updated by RFC 5462. Disponível em: <http://www.ietf.org/rfc/rfc3469.txt>. 40, 44, 78

SHERWOOD, R. et al. Flowvisor: A network virtualization layer. In: *Technical Report Openflow-tr-2009-1*. [S.l.]: Stanford University, 2009. 63

SHERWOOD, R. et al. Can the production network be the testbed? In: *Proceedings of the 9th USENIX conference on Operating systems design and implementation*. Berkeley, CA, USA: USENIX Association, 2010. (OSDI'10), p. 1–6. Disponível em: <http://dl.acm.org/citation.cfm?id=1924943.1924969>. 29

SUCHARA, M. et al. Network architecture for joint failure recovery and traffic engineering. In: *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM, 2011. (SIGMETRICS '11), p. 97–108. ISBN 978-1-4503-0814-4. Disponível em: <http://doi.acm.org/10.1145/1993744.1993756>. 65

TAVAKOLI, A. et al. Applying nox to the datacenter. 2009. october 2009. 17

TENNENHOUSE, D. L.; WETHERALL, D. J. Towards an active network architecture. *SIGCOMM Comput. Commun. Rev.*, 1996. ACM, New York, NY, USA, v. 26, n. 2, p. 5–17, abr. 1996. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/231699.231701>. 30

TOOTOONCHIAN, A.; GANJALI, Y. Hyperflow: a distributed control plane for openflow. In: *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. Berkeley, CA, USA: USENIX Association, 2010. (INM/WREN'10), p. 3–3. Disponível em: <http://dl.acm.org/citation.cfm?id=1863133.1863136>. 37, 65

TOP500. *Top500 Supercomputer Sites*. 2013. Último acesso: 06/06/2013. Disponível em: <http://www.top500.org/statistics/list/>. 14, 24

VASSEUR, J. P.; PACKAVET, M.; DEMEESTER, P. *Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS*. San Francisco, CA: Elsevier Inc, 2004. 14, 40, 41, 42, 43

VERDI, F. L. et al. Novas arquiteturas de data center para cloud computing. *SBRC 2010 - Minicursos*, 2010. Gramado, Brazil, maio 2010. 17, 23, 28

WMWare Staff. 2013. 2013. 23

YUAN, X. et al. Oblivious routing for fat-tree based system area networks with uncertain traffic demands. *SIGMETRICS Perform. Eval. Rev.*, 2007. ACM, New York, NY, USA, v. 35, n. 1, p. 337–348, jun. 2007. ISSN 0163-5999. Disponível em: <http://doi.acm.org/10.1145/1269899.1254922>. 61, 64

Publicações do Autor

1. RAMOS, R. M.; MARTINELLO, M.; Rothenberg. C. E. . Data Center Fault-Tolerant Routing and Forwarding: An Approach based on Encoded Paths. **LADC - Sixth Latin-American Symposium on Dependable Computing**, Rio de Janeiro, 2013.

2. RAMOS, R. M.; MARTINELLO, M.; Rothenberg. C. E. . SlickFlow: Resilient Source Routing in Data Center Networks Unlocked by OpenFlow. **LNC - The 38th IEEE Conference on Local Computer Networks**, Sydney, 2013.