UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA PROPOSTA DE REDIRECIONAMENTO DE FLUXOS DE REDE USANDO OPENFLOW PARA MIGRAÇÃO DE APLICAÇÕES ENTRE NUVENS

CARLOS SPINETTI MODA

ORIENTADOR: PROF. DR. HERMES SENGER

São Carlos – SP

Fevereiro/2014

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA PROPOSTA DE REDIRECIONAMENTO DE FLUXOS DE REDE USANDO OPENFLOW PARA MIGRAÇÃO DE APLICAÇÕES ENTRE NUVENS

CARLOS SPINETTI MODA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.
Orientador: Prof. Dr. Hermes Senger

São Carlos – SP

Fevereiro/2014

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

M689pr

Moda, Carlos Spinetti.
Uma proposta de redirecionamento de fluxos de rede usando *openflow* para migração de aplicações entre nuvens / Carlos Spinetti Moda. -- São Carlos : UFSCar, 2014.
93 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2014.

1. Redes de computação - protocolos. 2. Computação em nuvem. 3. Redes definidas por software. 4. *Infrastructure as a Service* (IaaS). 5. *OpenFlow*. 6. Migração de aplicações. I. Título.

CDD: 004.62 (20ª)

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

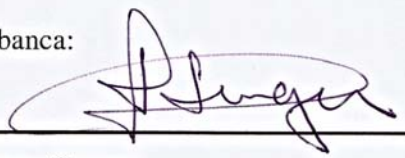
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA PROPOSTA DE REDIRECIONAMENTO DE FLUXOS DE REDE USANDO OPENFLOW PARA MIGRAÇÃO DE APLICAÇÕES ENTRE NUVENS

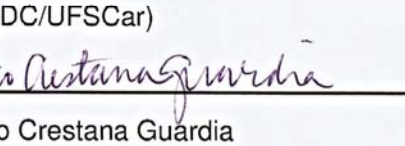
CARLOS SPINETTI MODA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da banca:



Prof. Dr. Hermes Senger
(Orientador– DC/UFSCar)



Prof. Dr. Hélio Crestana Guardia
(DC/UFSCar)

Profa. Dra. Líria Matsumoto Sato
(POLI/USP)

São Carlos – SP

Fevereiro/2014

À minha família

AGRADECIMENTOS

Agradeço à minha família, pelo apoio e pelo suporte durante todos esses anos. Agradeço ao Prof. Hermes Senger, pela orientação, pela paciência e pela motivação a fazer um trabalho sempre melhor. Agradeço também ao caro amigo Lucas Venezian Povoa, pela grande ajuda com a parte experimental e por todo o apoio, mesmo nas horas mais difíceis. Agradeço a todos os colegas e amigos do GSDR e do Asgard, pelo companheirismo e por tornarem mais leve o peso do trabalho.

"...Liberdade, essa palavra
que o sonho humano alimenta
que não há ninguém que explique
e ninguém que não entenda."

Cecília Meireles

RESUMO

Durante a última década, o advento do processamento em larga escala e a necessidade de rápida modificação de estruturas computacionais fez com que a computação em nuvem se popularizasse, em particular na forma de aprovisionamento de Infraestrutura como Serviço. Diversas companhias investiram em infraestrutura para se tornarem provedores desse tipo de serviço, seja para o público ou para proverem recursos para seus próprios negócios. Isto aumentou o número de centros de dados virtualizados e gerou o interesse na interoperabilidade entre os diferentes provedores. Entretanto, devido à falta de padronização de tecnologias, e devido a limitações na arquitetura das redes atuais, essa interoperabilidade ainda é um assunto em aberto. Com base nisso, o presente trabalho apresenta uma arquitetura de redirecionamento de fluxos de rede baseada em OpenFlow para o suporte à continuidade de serviço durante a migração de aplicações entre diferentes provedores de IaaS. Os testes realizados comprovam sua aplicabilidade em um cenário real, controlando apenas as bordas da rede, e sem a instalação de nenhum hardware específico.

Palavras-chave: Computação em nuvem, IaaS, Redes Definidas por Software, OpenFlow, Redirecionamento de fluxos, migração de aplicações

ABSTRACT

During the last decade, the advent of large scale processing and the need for rapid modification of computational structures have increased the popularity of Cloud Computing, particularly the Infrastructure as a Service model. Several companies have invested in infrastructure to become providers of this kind of service, whether for general public or only to supply their own business needs. This has increased the number of virtualized datacenters across the world and created a growing interest in interoperability between different providers. However, due to the lack of technology standardization, and to limitations in the current network's architecture, this interoperability is still an issue. Based on this, this research project presents an OpenFlow based network flow redirection architecture to support service continuity during the migration of applications between different IaaS providers. The tests performed show the applicability of the proposed architecture in a real network environment, having control only of the network edges, and without setting up any specific hardware.

Keywords: IaaS Cloud Computing, Software Defined Networks, OpenFlow, flow redirection, application migration

LISTA DE FIGURAS

2.1 Relação entre os modelos de entrega oferta de serviços de computação em nuvem. Adaptada de (SHALOM, 2012) 24

2.2 Arquitetura de uma Rede Definida por Software. Adaptado de (Open Networking Foundation, 2012). 28

2.3 Funcionamento de uma rede *OpenFlow*. Adaptado de McKeown et al. 29

2.4 Estrutura de campos de cabeçalho que caracteriza um fluxo *OpenFlow*. Adaptado de McKeown et al. 30

2.5 Entrada de uma tabela de fluxos. Adaptado de McKeown et al. 31

2.6 Extensão de rede local com VPNs. Adaptado de Hirofuchi et al. 34

2.7 Exemplos de VANs em um ou mais sítios. Adaptado de Hadas et al. 36

2.8 Detalhes do protocolo VAN. Adaptado de Hadas et al. 37

2.9 Esquema de tunelamento IP para migração de máquinas virtuais, adaptado de (TRAVOSTINO, 2006) 38

2.10 Arquitetura da plataforma VICTOR. Adaptado de Hao et al. 41

2.11 Operações de reescrita de cabeçalhos do *DCPortals*. Adaptado de Nunes et al. . 43

3.1 Representação lógica do funcionamento do *Switch* Virtual. 48

3.2 Visão geral da arquitetura proposta. 50

3.3 Exemplo de mecanismo para manter a consistência dos dados da aplicação através de serviço de *Block Storage* 52

3.4 Adaptação do mecanismo de Hirofuchi et al. Os dados da aplicação são armazenados em outra MV e montados pelas MVs que executam a aplicação. Essa estrutura de duas MVs é replicada no sítio destino, e as MVs que armazenam os dados se comunicam entre si através do mecanismo e operações de sincronização de dados de disco para manterem a consistência dos dados da aplicação. 52

3.5 Estado inicial: a origem atende as requisições diretamente. 54

3.6 Comandos enviados pelo gerente de migração para iniciar o redirecionamento de fluxos. 55

3.7 Comunicação entre os elementos da arquitetura na 1ª versão do redirecionamento de fluxos. 57

3.8 Estrutura de dados que identifica os fluxos dos clientes. 57

3.9 Substituição de campos e encaminhamento de pacotes na primeira versão do redirecionamento de fluxos. As setas indicam o encaminhamento do fluxo, e as estruturas mostram os campos de cabeçalhos principais referentes aos fluxos. . 58

3.10 Comunicação entre agentes e reescrita de pacotes na segunda versão do redirecionamento de fluxos. 60

3.11 Estado final: o destino passa a atender as requisições diretamente. 61

4.1 Topologia do cenário de testes com nuvens privadas. 68

4.2 Gráfico do tempo de resposta do teste realizado em ambiente de núvem privada. 68

4.3 Topologia do cenário utilizado nos testes de redirecionamento entre Ourinhos e AWS. 69

4.4 Gráficos de distribuição das cargas sintéticas geradas. 74

4.5 Gráficos de análise de dispersão dos tempos de resposta. 75

4.6 Gráficos de distribuição dos tempos de resposta. 76

4.7 Ganho no tempo de resposta comparado com o servidor original. 76

4.8 Perda de pacotes durante todo o processo. 77

LISTA DE TABELAS

2.1 Quadro comparativo dos trabalhos sobre migração e continuidade de serviço. . 39

4.1 Configuração das instâncias utilizadas para Destino 70

4.2 *Softwares* instalados nas máquinas do ambiente de testes. 71

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO **14**

1.1 Motivação 16

1.2 Justificativa para o uso do OpenFlow 17

1.3 Objetivos 18

1.4 Organização da dissertação 19

CAPÍTULO 2 – CONCEITOS RELACIONADOS AO TRABALHO **20**

2.1 Computação em Nuvem 20

2.1.1 Definição 21

2.1.1.1 Características essenciais 21

2.1.1.2 Ofertas de serviços 22

2.1.1.3 Modelos de implantação 23

2.1.2 *Amazon Web Services* 25

2.2 Redes Definidas por Software 25

2.3 *OpenFlow* 27

2.3.1 Elementos de uma rede *OpenFlow* 28

2.3.2 Tabela de fluxos 29

2.3.3 Funcionamento de uma rede *OpenFlow* 31

2.3.4 Sistema operacional de rede 32

2.4 Migração e continuidade de serviço 32

2.4.1 Comparação entre os trabalhos 39
2.5 Redirecionamento de fluxos com *OpenFlow* 40
2.5.1 Pontos a destacar 43
2.6 Conclusões 43

CAPÍTULO 3 – PROPOSTA DE ARQUITETURA PARA REDIRECIONAMENTO DE FLUXOS **45**

3.1 Projeto da arquitetura 45
3.2 Objetivos da Proposta 47
3.3 Componentes da arquitetura proposta 47
3.3.1 *Switch* Virtual 48
3.3.2 Controlador *OpenFlow* 48
3.3.3 DNS Dinâmico (cliente e servidor) 49
3.3.4 Gerente do redirecionamento 49
3.4 Visão geral da arquitetura 50
3.5 Integração com outros mecanismos 51
3.6 Exemplo do funcionamento da arquitetura em uma migração de aplicação . . . 53
3.6.1 Estado inicial 53
3.6.2 Início da migração 54
3.6.3 Redirecionamento de fluxos 55
3.6.3.1 Primeira versão 56
3.6.3.1.1 O problema dos fluxos novos 56
3.6.3.2 Segunda versão 58
3.6.4 Final da migração 60
3.6.5 Comentários sobre as duas versões do redirecionamento 60
3.7 Implementação do protótipo funcional 62
3.7.1 Controlador *OpenFlow* 62

3.7.1.1 Eventos 62
3.7.1.2 Estruturas de dados internas 64
3.7.1.3 Rede interna dos provedores 65
3.7.2 *Switch* virtual e DNS dinâmico 65

CAPÍTULO 4 – TESTE DA ARQUITETURA **66**

4.1 Teste da 1ª versão do redirecionamento de fluxos 66
4.2 Testes da 2ª versão do redirecionamento de fluxos 67
4.2.1 Teste com nuvens privadas 67
4.2.1.1 Resultados 67
4.2.2 Testes com nuvem pública 69
4.2.2.1 Configuração de *hardware* 70
4.2.2.2 Configuração de *software* 70
4.2.2.3 Configuração de rede 71
4.2.2.4 Testes de funcionalidade 71
4.2.2.5 Testes de desempenho 72
4.2.2.5.1 Geração da carga de trabalho 72
4.2.2.5.2 Descrição dos testes 73
4.2.2.6 Análise dos resultados 74

CAPÍTULO 5 – CONCLUSÕES E TRABALHOS FUTUROS **78**

5.1 Conclusões 78
5.2 Trabalhos Futuros 79

APÊNDICE A – CÓDIGOS-FONTE E *SCRIPTS* **80**

REFERÊNCIAS **88**

GLOSSÁRIO **92**

Capítulo 1

INTRODUÇÃO

A computação em nuvem está transformando a indústria da Tecnologia da Informação e Comunicação (T.I.C.). O processamento e o armazenamento dos dados, antes realizados por *laptops*, pelos computadores pessoais ou pequenos servidores corporativos passou a ser realizado em grandes *datacenters* (DIKAIAKOS, 2009) de propriedade dos próprios usuários ou empresas, ou ainda de terceiros, que vendem ou alugam recursos para quem necessite. A computação em nuvem criou um mercado onde os *provedores* são aqueles que adquirem milhares de recursos tais como servidores, discos e equipamentos de rede, e vendem suas capacidades para outros usuários ou empresas, denominados *clientes*. A ideia central nesse modelo é disponibilizar recursos de processamento, armazenamento de dados e *software* e entregá-los através da rede, sob a forma de serviços. Os clientes podem contratar tais serviços de maneira fácil e rápida, pagando apenas um valor proporcional ao uso. Esse modelo é conhecido como *"pay-as-you-go"*, e funciona de forma similar a serviços como água encanada e energia elétrica. (BUYYA; RANJAN; CALHEIROS,)

Apesar da enorme variedade de serviços e recursos que podem ser oferecidos e contratados por meio da computação em nuvem, este trabalho se insere no contexto do aprovisionamento de infraestrutura como serviço (*Infrastructure-as-a-Service - IaaS*). Nesta modalidade, os provedores ofertam recursos de computação, geralmente entregues na forma de máquinas virtuais (MVs) que podem ser configuradas para executar aplicações dos clientes. Também são comuns serviços de armazenamento de blocos de dados, armazenamento de objetos, balanceamento de carga, entre outros. Comumente, os clientes podem ser provedores de outros serviços, tais como e-mail, distribuição de vídeos e redes sociais. Empresas como Facebook, Yahoo e Twitter são exemplos de consumidores de recursos de computação em nuvem que são também provedores de outros serviços, que são oferecidos, por sua vez, aos *usuários finais*. Esse modelo tem permitido que empresas pequenas comecem novos negócios com pouco capital inicial e reduzidos

investimentos em infraestrutura, e que cresçam rapidamente, acomodando seus custos de acordo com o aumento de suas vendas. Empresas de maior porte, ou que tenham demanda suficiente, podem adquirir suas próprias infraestruturas de nuvem, para atender à demanda interna. Nesses casos, dá-se o nome de nuvem privada.

Entre as vantagens do aprovisionamento de infraestrutura como serviço, podemos citar a grande redução dos investimentos em aquisição e manutenção da infraestrutura computacional para os clientes, visto que esta é mantida pelos provedores. Os clientes podem alocar ou desalocar os recursos de que necessitam, de forma simples e rápida, para acompanhar o crescimento ou a redução do volume de seus negócios, e para acomodar picos de demanda. Além disso, só pagam pelo que realmente utilizam, evitando assim desperdício com eventual superdimensionamento dos recursos. Do outro lado, os provedores se beneficiam com a economia de escala. Tecnologias como a virtualização permitem aumentar o grau de compartilhamento de seus recursos físicos, diminuindo os gastos com manutenção e atualização de *software*, além de otimizarem a utilização dos mesmos, em termos de consumo de energia elétrica e refrigeração. Essa economia se reflete na redução dos preços para o consumidor.

Todos esses benefícios têm motivado empresas a utilizarem cada vez mais a computação em nuvem. Além disso, na última década, surgiram diversas empresas atuando como provedores de infraestrutura como serviço, tais como Amazon, Rackspace, GoGrid, Terremark, HP, Microsoft, IBM, entre outras. Tais provedores oferecem diferentes políticas de preços e formas de contratação de recursos, bem como diferentes níveis de qualidade de serviço e disponibilidade, o que faz com que clientes queiram ter a liberdade de escolher quais provedores irão contratar, e por quanto tempo. Para isto, eles precisam ter meios de mover suas aplicações de um provedor para outro. Essa mobilidade, entretanto, não é algo trivial.

A mobilidade de aplicações entre ambientes de IaaS é algo complexo e possui uma série de aspectos precisam ser levados em conta. Nos casos de aplicações que já executam em uma infraestrutura de nuvem, há casos em que estas necessitam ser movidas, seja de um provedor de IaaS público para outro, ou para uma infraestrutura de nuvem privada. Essa mobilidade entre provedores envolve uma série de pontos a serem verificados para garantir a mobilidade da aplicação, como compatibilidade entre as plataformas de origem e de destino, compatibilidade entre as tecnologias de virtualização utilizadas na origem e no destino, compatibilidade entre os mecanismos de gerenciamento de recursos entre origem e destino e conectividade dos clientes com a aplicação em movimento.

Dentre todos os aspectos envolvidos na mobilidade de aplicações, talvez o mais crítico seja a continuidade do serviço. Em geral, para mover aplicações de uma infraestrutura à outra

se faz necessário interromper o serviço por um determinado intervalo de tempo para concluir o processo. Em um mercado competitivo e globalizado, com uma necessidade crescente de operação ininterrupta dos serviços, e as antigas ”janelas de manutenção” sendo reduzidas ao máximo, ou até eliminadas, a interrupção do serviços torna-se intolerável.

1.1 Motivação

A mobilidade de aplicações sem interrupção de serviço é um tópico de grande interesse, tanto na indústria quanto na academia. Na indústria, atualmente, é possível encontrar algumas soluções para a mobilidade de aplicações, entretanto, essas são proprietárias, e, por isso, limitadas a tecnologias e plataformas específicas (AppZero, 2013; NETIQ CORPORATION, 2014). Na literatura, encontram-se diversos trabalhos relacionados a mover aplicações para a nuvem, mas esses, em sua maioria, concentram-se em aperfeiçoar processos de refatoração de aplicações a serem movidas pela primeira vez para a nuvem, e não em evitar a interrupção de serviço durante a movimentação da aplicação (BANERJEE, 2012; CHAUHAN; BABAR, 2012). Os trabalhos que abordam a mobilidade de aplicações no contexto de IaaS e que visam a manter a continuidade da aplicação, em sua maioria, são dedicados à migração de máquinas virtuais. Isso ocorre pois a virtualização é a tecnologia subjacente predominante nesse cenário. Muitos desses trabalhos propõem diferentes estratégias de migração, que têm como objetivo migrar uma imagem completa de uma máquina virtual (com todas as aplicações e dados) sem que as aplicações em execução dentro dela sejam interrompidas sob o ponto de vista do usuário.

A migração de máquinas virtuais tem sido bastante estudada nos últimos anos, sendo tratada em diferentes cenários. Em ambientes de múltiplos servidores, executando um mesmo hipervisor, e confinados em uma mesma rede local, a migração sem interrupção de serviço é uma questão praticamente resolvida pelos fabricantes e fornecedores de hipervisores, como *VMWare* e *Citrix*, sendo inclusive chamada de migração ao vivo (*live migration*). A migração de máquinas virtuais entre ambientes de IaaS constitui uma questão mais complexa, visto que os servidores não mais se encontram em uma mesma rede local, e os hipervisores utilizados na origem e no destino podem ser diferentes. Além disso, existem outras complicações relacionadas à compatibilidade entre os componentes de *software* utilizados para gerenciamento da infraestrutura de nuvem, os chamados controladores de nuvem, tais como *Eucalyptus*, *OpenStack*, *CloudStack*, entre outros.

A análise de trabalhos envolvendo a migração de máquinas virtuais entre redes de longa distância (*Wide Area Networks*, ou WANs) identificou várias técnicas e estratégias para oti-

mizar a transferência de dados através dessas redes, especificamente, grandes quantidades de dados, correspondentes a discos virtuais de MVs. Há técnicas interessantes que poderiam ser adaptadas para o contexto da migração de aplicações. Apesar disso, notou-se nesses trabalhos que a manutenção da conectividade e da continuidade de serviço ao se realizar uma migração através de WANs é abordada de forma bastante restrita. Nesse ponto, foram identificados apenas alguns mecanismos para manter essa conectividade, todos baseados em alguma forma de encapsulamento de pacotes, como tunelamento IP sobre IP, VPNs ou redes de sobreposição (*overlay networks*). Dessa forma, este trabalho concentra-se nesses aspectos de rede, buscando aprimorar o suporte à continuidade de serviço durante a migração de aplicações no contexto de ambientes de IaaS.

1.2 Justificativa para o uso do OpenFlow

A maioria dos trabalhos que tratam da mobilidade de aplicações não trata do redirecionamento de fluxos, ou o faz de maneira restrita. Possivelmente, isso se deve à limitação dos protocolos atuais de rede utilizados na Internet, o que dificulta a inovação. Essa dificuldade é relatada por diversos pesquisadores da área de Redes de Computadores, e é conhecida como ”engessamento”ou ”ossificação”da Internet. Entre as alternativas que vêm sendo criadas para contornar esse problema destacam-se as Redes Definidas por Software (RDS), e sua implementação mais difundida, o protocolo *OpenFlow*. Através de pequenas modificações nos equipamentos de rede atuais, e de uma forma inovadora de se comunicar com eles, *OpenFlow* garante um controle muito maior e mais fácil de redes, além de garantir maior flexibilidade das mesmas, dando aos administradores de redes maior liberdade de configuração e manutenção. Desde sua criação, em 2008, o *OpenFlow* tem ganhado importância significativa no cenário atual, sendo cada vez mais utilizado por diversas instituições de pesquisa e empresas. Um exemplo que pode ser citado para ilustrar seu destaque nos dias de hoje é o anúncio do Google de que o está utilizando para a interligação de seus *datacenters* (LEVY, 2010). *OpenFlow* é um protocolo aberto, e tornou-se o primeiro padrão para RDS (Open Networking Foundation, 2012). Em decorrência dessa grande visibilidade, vários fabricantes de equipamentos de rede já o incluíram em seus produtos, que estão se tornando amplamente disponíveis.

Partindo dessa motivação, foram analisados trabalhos que utilizaram *OpenFlow* para a manipulação de fluxos de redes em diferentes contextos, com diferentes finalidades. Os resultados desses trabalhos mostraram a eficiência dessa tecnologia na manipulação de fluxos, superando outros métodos, como os baseados em empilhamento de protocolos. Isso ocorreu devido a uma característica do *OpenFlow* que permite que campos de cabeçalhos de pacotes

sejam reescritos conforme a conveniência. Essa análise também revelou que é possível utilizar soluções baseadas em *OpenFlow* mesmo sem possuir equipamentos de rede que suportem seu protocolo. É possível realizar implementações utilizando apenas elementos de *software*, como comutadores (*switches*) virtuais que suportam *OpenFlow*, apenas nas bordas da rede, e ainda sim obter ganhos de desempenho. Essa informação é particularmente importante para o contexto deste trabalho, pois ao utilizar o modelo IaaS geralmente não se tem o controle dos recursos físicos (com exceção dos casos de nuvens privadas). Logo, uma boa solução para a mobilidade de aplicações nesse contexto necessita ser independente da infraestrutura subjacente. Acredita-se, entretanto, que com a grande visibilidade que o *OpenFlow* vem ganhando, é possível que num futuro próximo, provedores de IaaS passem a oferecer serviços de rede baseados nessa tecnologia, o que tornaria a abordagem deste trabalho mais abrangente e eficiente.

1.3 Objetivos

Seguindo a motivação apresentada por esse contexto, são traçados os seguintes objetivos para este trabalho:

- Propor uma arquitetura para redirecionamento de fluxos de rede visando o suporte à mobilidade de aplicações no contexto do aprovisionamento de infraestrutura como serviço, tendo como base conceitos de Redes Definidas por *Software*;
- Validar a hipótese de que os conceitos de RDS podem oferecer uma solução simples e barata para o redirecionamento de fluxos de rede em ambientes de computação em nuvem, sem a necessidade de equipamento específico ou alterações nas configurações dos ambientes;
- Manter a conectividade de rede dos usuários finais com a aplicação durante o processo de migração, procurando assim minimizar o impacto do processo na qualidade de serviço;
- Propor uma arquitetura flexível que possa ser empregada em diversos casos dentro do cenário do trabalho, através da utilização de padrões tecnológicos abertos e de uma proposta independente de equipamentos e de infraestrutura de provedores;
- Fornecer um protótipo funcional que ilustre a implementação da arquitetura proposta.

1.4 Organização da dissertação

O restante deste documento está organizado da forma descrita a seguir. O Capítulo 2 apresenta conceitos relacionados ao trabalho, bem como a análise de alguns trabalhos relacionados ao tema, procurando expor o problema a ser abordado e justificar a proposta do trabalho. O Capítulo 3 descreve a arquitetura proposta, seus componentes, alguns cenários em que pode ser usada e apresenta os detalhes referentes à implementação do protótipo funcional. O Capítulo 4 descreve os experimentos realizados para testar a arquitetura, exibe os resultados obtidos e os analisa. Por fim, o Capítulo 5 apresenta as conclusões obtidas com o desenvolvimento do trabalho e aponta para trabalhos futuros.

Capítulo 2

CONCEITOS RELACIONADOS AO TRABALHO

Neste capítulo são apresentados alguns conceitos relacionados ao trabalho, visando o entendimento de seu contexto, bem como conceitos necessários ao entendimento da arquitetura proposta. A seguir são analisados alguns trabalhos relacionados aos temas abordados pelo trabalho, tais como a manutenção de continuidade de serviços durante a migração de aplicações em ambientes de computação em nuvem e o redirecionamento de fluxos de rede baseado Redes Definidas por *Software*.

2.1 Computação em Nuvem

Atualmente, a computação em nuvem é um assunto muito popular. Muito tem se falado a respeito desse modelo e de suas vantagens em relação a outros adotados até então. Todo esse destaque acaba criando a impressão de se tratar de uma ideia recente. Essa visão, entretanto, é equivocada. Apesar de sua adoção em larga escala ser de fato recente, as ideias principais por trás da computação em nuvem são antigas.

O conceito de entrega de serviços computacionais através uma rede global remonta aos anos de 1960. Especialistas atribuem um dos principais conceitos por trás da computação em nuvem a John McCarty, que em seu discurso no Centenário do MIT, em 1961, disse que ”a computação pode um dia ser organizada como uma utilidade pública”(GARFINKEL; ABELSON, 1999). Em 1966, em seu livro *”The challenge of the computer utility”*, Douglas Parkhill explora quase todas as características da computação em nuvem dos dias de hoje (PARKHILL, 1966).

Muitos esforços, reunidos ao longo de décadas, tentando criar meios de proporcionar computação como utilidade em larga escala foram frustrados devido a limitações nas redes de telecomunicações da época, como o sistema telefônico. Dessa forma, a solução mais viável e barata para com-

panhias e outras organizações era armazenar dados e executar aplicações em sistemas computacionais privados, mantidos em suas próprias instalações físicas. Foi apenas em meados dos anos de 1990 que a capacidade das redes começou a aumentar a ponto de viabilizar esse tipo de serviço. Nessa época, com o rápido crescimento da Internet, as empresas de telecomunicações passaram a investir em redes de fibra óptica de alta capacidade.

Ao final dessa década, diversas companhias, conhecidas como provedores de serviços, foram fundadas com o objetivo de prover aplicações computacionais a outras companhias através da Internet. Muitos desses pioneiros faliram, porém seu modelo de negócio acabou se popularizando uma década mais tarde, com a computação em nuvem.

2.1.1 Definição

Acredita-se que o termo ”Computação em Nuvem”tenha sido utilizado pela primeira vez, no contexto de entrega de serviços de computação pela Internet, em 2006 por Erick Schimidt, na época CEO do Google (AYMERICH; FENU; SURCIS,). Desde então, esse termo vem sido repetido exaustivamente, e devido ao grande marketing em torno do assunto, acabou sendo empregado em diferentes contextos, para representar diferentes conceitos. Isso acabou gerando uma grande confusão no mercado, que trouxe consigo ceticismo e insegurança acerca da computação em nuvem. Por essa razão, esforços foram empregados para se atingir uma padronização na definição de computação em nuvem. Entre as definições mais aceitas e referenciadas está a do Instituto Nacional de Padrões e Tecnologia dos Estados Unidos (*National Institute of Standards and Technology* - NIST) (MELL; GRANCE, 2011), uma instituição mundialmente reconhecida, que reuniu de forma concisa e objetiva as principais características desse modelo:

> “Computação em nuvem é um modelo para habilitar o acesso ubíquo, conveniente e sob demanda a um conjunto de recursos computacionais (como redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente providos e liberados com esforço mínimo de gerenciamento e interação com o provedor.”

Além dessa definição, o NIST propõe um modelo de referência para a computação em nuvem que considera suas características essenciais, ofertas de serviços e modelos de implementação mais comuns. Nos itens a seguir são exibidas resumidamente cada uma dessas dimensões.

2.1.1.1 Características essenciais

Segundo o NIST, o paradigma de computação em nuvem apresenta cinco características consideradas essenciais: auto-serviço sob demanda, acesso amplo através da rede, agrupamento

de recursos, rápida elasticidade e capacidade de mensurar os serviços oferecidos.

- *Auto-serviço sob demanda*: dentro do modelo de computação em nuvem os clientes têm o poder de alterar o provisionamento das capacidades computacionais que adquiriram no momento em que lhes for conveniente, e isso é feito sem a necessidade de interação humana com o provedor de serviço.

- Acesso amplo através da rede: as capacidades computacionais estão disponíveis através da rede e podem ser acessadas via mecanismos padronizados, a partir de diferentes plataformas, como *desktops* e *laptops*, celulares, *smartphones*, *tablets*, entre outros.

- Agrupamento (*pooling*) de recursos: os recursos computacionais do provedor de serviço são agrupados e dispostos de modo a atenderem múltiplos clientes. Diferentes recursos físicos e virtuais podem ser dinamicamente atribuídos e reatribuídos de acordo com a demanda de cada consumidor. Os recursos providos são independentes de suas localizações geográficas, porém deve haver a possibilidade dos clientes especificarem essa localização em um nível mais alto de abstração (como país, estado ou *datacenter*). Exemplos de recursos incluem armazenamento, processamento, memória e largura de banda.

- Rápida elasticidade: os recursos contratados podem ser alocados e desalocados elasticamente, de modo que podem ser rapidamente escalados para mais ou para menos, proporcionalmente à demanda dos clientes. Em geral, sob o ponto de vista do consumidor, as capacidades disponíveis para aprovisionamento parecem ilimitadas, pois podem ser alocadas em qualquer quantidade, a qualquer momento.

- Medição de serviço: sistemas de computação em nuvem automaticamente controlam e otimizam o uso de recursos através de serviços de medições, em um determinado nível de abstração, adequado ao tipo de serviço contratado (como armazenamento, processamento, largura de banda, etc.). O uso dos recursos pode ser monitorado, controlado e relatado, provendo transparência do serviço utilizado, tanto para o provedor quanto para o consumidor.

2.1.1.2 Ofertas de serviços

Dentro da computação em nuvem se encaixam três modelos básicos de oferta de serviços através da Internet. Esses modelos são chamados de Software como serviço, Plataforma como serviço e Infraestrutura como serviço, explicados a seguir. A figura 2.1 ilustra os diferentes

modelos, seus respectivos consumidores usuais e o modo como os modelos de oferta são relacionados entre si.

- Software como Serviço (*Software as a Service* - SaaS): nesse modelo o consumidor é um usuário final de aplicações completas, que são executadas diretamente na nuvem. Essas aplicações são tipicamente acessadas através de uma interface leve, como um navegador de Internet. No modelo SaaS o consumidor não possui nenhum controle sobre quaisquer elementos subjacentes às aplicações que está executando.

- Plataforma como Serviço (*Platform as a Service* - PaaS): nesse modelo o consumidor tipicamente é um desenvolvedor de *software*. Aqui o provedor oferece um conjunto de ferramentas e um número de linguagens de programação suportadas para que o cliente possa desenvolver aplicações que acessam e utilizam recursos da nuvem.

- Infraestrutura como Serviço (*Infrastructure as a Service* - IaaS): nesse modelo são oferecidos recursos de infraestrutura básicos, como capacidade de processamento, áreas de armazenagem, entre outros. O consumidor não tem controle sobre parâmetros da infraestrutura física, porém pode instalar um sistema operacional de sua escolha, bem como quaisquer *softwares* sobre ele. Em outras palavras, ele tem controle total a partir da inicialização (*boot*) do sistema. Por oferecer maior liberdade e maior controle sobre as capacidades adquiridas, o modelo IaaS é o mais utilizado por companhias que já provêm serviços a usuários finais e desejam se beneficiar das vantagens da computação em nuvem. Também por esse motivo, o modelo IaaS é o foco deste trabalho.

2.1.1.3 Modelos de implantação

Por fim, o NIST cita em sua definição quatro modelos de implantação para infraestruturas de computação em nuvem: Nuvens privadas, nuvens públicas, nuvens híbridas e nuvens comunitárias. A seguir são definidos cada um deles.

- Nuvens privadas: são aquelas em que a infraestrutura é montada para uso exclusivo de uma única organização, que é dona da nuvem e tem controle total sobre ela. Uma nuvem privada pode pertencer a uma organização mas ser montada, instalada e gerenciada por terceiros. A infraestrutura pode ser instalada ou não em um local de propriedade dessa organização.

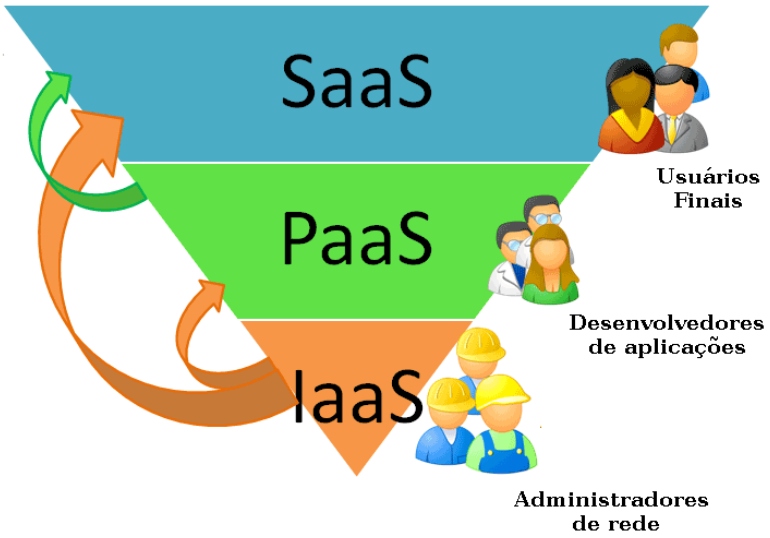


Figura 2.1: Relação entre os modelos de entrega oferta de serviços de computação em nuvem. Adaptada de (SHALOM, 2012)

- Nuvens públicas: são aquelas que pertencem a um provedor de serviços de nuvem. A infraestrutura de nuvem é disponibilizada para uso aberto pelo público em geral, que paga pela utilização dos recursos. Elas podem pertencer, serem gerenciadas e operadas por companhias privadas, organizações acadêmicas, organizações governamentais, ou uma combinação dos mesmos.

- Nuvens comunitárias: são aquelas em que a infraestrutura de nuvem é montada para uso exclusivo de uma comunidade específica de usuários de organizações que possuem interesses em comum (como missões, requisitos de segurança, políticas e considerações de conformidade). Elas podem pertencer, serem gerenciadas e operadas por uma ou mais organizações da comunidade, por terceiros, ou por uma combinação dos mesmos.

- Nuvens híbridas: são aquelas cuja infraestrutura (de nuvem) é uma composição de duas ou mais nuvens distintas (privada, pública ou comunitária), que permanecem entidades únicas, mas que estão unidas por tecnologias padronizadas ou proprietárias, que permitem a portabilidade de dados e de aplicações.

2.1.2 *Amazon Web Services*

Um dos primeiros provedores de IaaS, que até hoje se mantem como um dos principais no mercado é a Amazon. Atualmente, sua linha *Amazon Web Services* (AWS) (Amazon.com, Inc., 2014c) oferece um grande número de serviços na nuvem, como armazenamento de dados, DNS escalável, balanceamento de carga, bancos de dados, entre outros. Um desses serviços, o *Elastic Compute Cloud* (EC2) (Amazon.com, Inc., 2014b), segue os moldes de provisionamento IaaS através da oferta de máquinas virtuais sob demanda. As máquinas virtuais, ou *instâncias*, podem ser criadas de maneira rápida e fácil pelos próprios clientes, através de uma interface *web*. Para a criação de instâncias a AWS disponibiliza recursos de diversos *datacenters*, ou *Availability Zones*, abstraindo a localização dos mesmos[1]. Há a capacidade de alocar mais ou menos instâncias de acordo com a necessidade do cliente, com algumas limitações. A AWS oferece aos seus clientes um *Service Level Agreement* (SLA) para garantir um nível mínimo de qualidade de serviço (*Quality of Service*, ou QoS). O SLA da AWS estipula um nível de 99,95% de disponibilidade de serviço por ano, além de outros aspectos de segurança, confiabilidade e escalabilidade.

Há vários modelos de pagamento pela contratação dos serviços, sendo que o mais utilizado é o *on-demand*, onde as MVs são instanciadas sob demanda e são tarifadas por hora de uso. Diversos tipos de instâncias são oferecidos pelo AWS EC2, separados por famílias, de acordo com a configuração de cada tipo. Há instâncias com mais memória RAM, com mais processamento, com mais armazenamento, com GPUs, entre outras. Entretanto, por serem MVs, a especificação de *hardware* de todas elas nem sempre está disponível de maneira clara. O poder computacional das instâncias, por exemplo, é medido em ECUs (*EC2 Compute Units*), sendo que, segundo o sítio do EC2, um ECU equivale a um processador AMD Opteron 2007 ou um Intel Xeon 2007 de 1.0-1.2 GHz.

2.2 Redes Definidas por Software

A Internet é a aplicação computacional mais bem sucedida até hoje. Ela literalmente mudou a forma como as pessoas trabalham, se comunicam, se divertem, e hoje é um componente essencial de nossa sociedade. Sua arquitetura, projetada há quase 40 anos, se mostrou robusta e flexível o suficiente para acomodar seu assombroso crescimento e sua utilização cada vez mais ampla. Tamanho sucesso, entretanto, trouxe consigo consequências negativas. Apesar do sucesso da Internet se dever à sua arquitetura em camadas, que promove funcionalidade

[1] É possível, entretanto, escolher em qual *Availability Zone* se deseja criar as instâncias.

básica para uma grande variedade de aplicações, seu modelo de entrega por melhor esforço não se encaixa bem em todos os casos. O crescimento do escopo das aplicações que executam nela mostrou que sua arquitetura apresenta deficiências graves, tais como segurança, controle e estabilidade de roteamento, qualidade de serviço, entre outras.

A solução para tais problemas depende de mudanças na arquitetura da Internet. Isso, entretanto, tornou-se muito difícil no cenário atual. Os grandes interesses econômicos atraídos pela Internet, somados à sua natureza decentralizada fez com que grandes investimentos fossem feitos nessa área e que uma forte indústria surgisse em torno dela. Alterações estruturais desse tipo requerem que diferentes partes interessadas (*stakeholders*) entrem em um consenso para que as modificações sejam implementadas em escala global. Criar um consenso entre competidores no mercado global é praticamente impossível, e a isso se soma o fato de que um consenso eliminaria eventuais vantagens competitivas, visto que uma boa parte dos equipamentos de rede hoje em dia são implementados de modo fechado, com *softare* e *hardware* proprietários. O resultado é a enorme estagnação arquitetural dentro da Internet. Ao passo em que são desenvolvidos novos protocolos de alto nível e novas e melhores tecnologias de transmissão de dados, a arquitetura do núcleo da Internet permanece intocável. Isso ocorre de tal forma que até mesmo a implantação incremental das mais básicas e necessárias mudanças, como o IPv6, é dolorosamente lenta. Esse fenômeno tem sido chamado por especialistas e pesquisadores de "ossificação" ou "engessamento" da Internet (ANDERSON, 2005; TURNER; TAYLOR,).

Apesar dessas barreiras desencorajadoras, muita pesquisa tem sido desenvolvida buscando inovações que possam superar esse problema. Uma dessas inovações é a chamada Rede Definida por *Software* (RDS), que tem recebido grande destaque, principalmente pela boa recepção da indústria. Poucos anos após as primeiras publicações a esse respeito, já se vê conceitos de RDS sendo aplicados em redes de grandes instituições de pesquisa e de grandes corporações (LEVY, 2010). Esse grande interesse por parte da indústria estimulou a criação da *Open Network Foundation* (ONF), uma organização formada por membros da área acadêmica e de grandes empresas de tecnologia, como Google, Facebook, Microsoft, Deutshe Telekon, entre outras, visando a padronização e a promoção de tecnologias em Redes Definidas por *Software*. RDS é uma nova abordagem para o projeto de redes de computadores, baseada em abstração de funcionalidades. De acordo com definição apresentada pela ONF (Open Networking Foundation, 2012), em RDS ocorre a separação entre os sistemas de tomada de decisão a respeito de roteamento do tráfego (chamado de plano de controle) e os sistemas de encaminhamento do tráfego (chamado de plano de dados). Consequentemente, RDS define também interfaces padronizadas para a comunicação entre os dois planos.

O plano de controle é centralizado e composto por elementos de *software*, chamados de controladores. Os controladores mantêm uma visão global da rede. De posse dessa visão global e centralizada da rede, os controladores então implementam as diferentes funcionalidades de controle, como roteamento, *multicast*, segurança, controle de acesso, entre outras. O plano de controle permite que diversos controladores, implementando diferentes funcionalidades, executem simultaneamente. Dessa forma, em uma mesma RDS podem estar presentes várias funcionalidades, de maneira oposta ao que ocorre, por exemplo, nas redes TCP/IP convencionais, onde apenas uma pilha de protocolos é executada. Os controladores tomam as decisões de encaminhamento de pacotes baseados nas funcionalidades que implementam, e enviam instruções ao plano de dados. Essa comunicação se dá através de uma interface padronizada, que, de acordo com a definição da ONF, deve ser independente de fabricantes.

O plano de dados é composto por equipamentos de rede, como *switches* e roteadores, porém com funcionalidade reduzida. Ao invés de implementarem uma pilha de protocolos complexa como ocorre nas redes convencionais, os equipamentos em RDS necessitam apenas se comunicar com os controladores, receber instruções deles e executar as ações de encaminhamento baseadas nessas instruções. Isso, além de facilitar o projeto e a fabricação desses equipamentos, faz com que a alteração nas funcionalidades de controle independa dos equipamentos do plano de dados; desde que implemente a interface de comunicação com os controladores, o mesmo equipamento pode ser usado em qualquer RDS, executando qualquer funcionalidade.

O fato de serem elementos de *software* facilita a implementação dos controladores. Dessa forma, novos controladores, com novas funcionalidades ou com versões aprimoradas de funcionalidades existentes, podem ser desenvolvidos rapidamente, e de maneira independente dos fabricantes de equipamentos de rede. A centralização do plano de controle facilita a implantação de novos controladores nas RDS. Dessa forma, as redes se tornam muito mais flexíveis. Essas duas características combinadas abrem espaço para novas possibilidades, como a definição de interfaces padronizadas entre o plano de controle e as aplicações de redes, como ilustrado na Figura 2.2. Isso permite que as próprias aplicações alterem dinamicamente o comportamento da rede, conforme suas necessidades.

2.3 *OpenFlow*

Resultado de anos de pesquisa nessa área, *OpenFlow* foi a primeira tecnologia capaz de realizar uma separação entre os planos de controle e de dados em equipamentos de rede convencionais. Atualmente a maioria das implementações de RDS é feita através do *OpenFlow*,

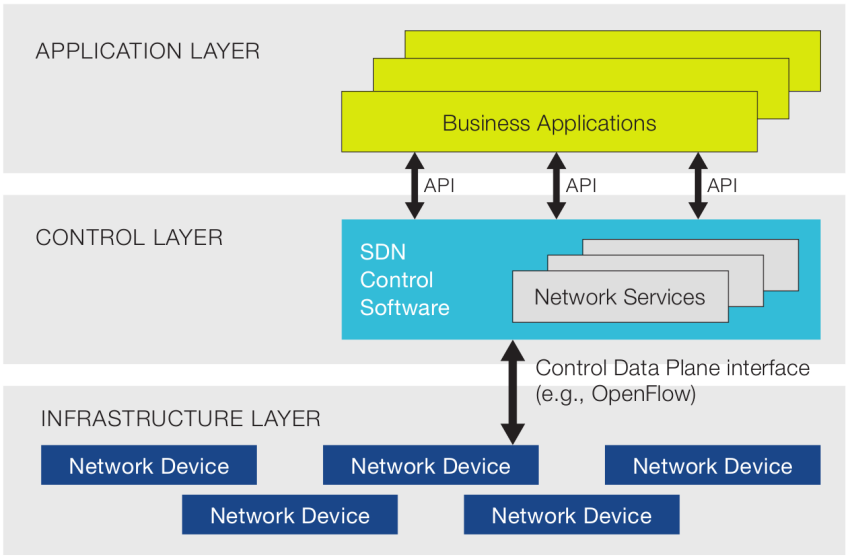


Figura 2.2: Arquitetura de uma Rede Definida por Software. Adaptado de (Open Networking Foundation, 2012).

e ele se tornou o primeiro protocolo padronizado para RDS (Open Networking Foundation, 2012). Inicialmente proposto por McKeown et al., (McKeown, 2008) *OpenFlow* surgiu como um meio de habilitar a pesquisa na área de Redes de Computadores. Com ele pesquisadores poderiam conduzir experimentos utilizando a infraestrutura de seus campus sem interferir na rede de produção. Para atingir esse objetivo, *OpenFlow* explora um recurso presente na maioria dos roteadores e *switches Ethernet* modernos, as tabelas de fluxo. Essas tabelas, geralmente construídas com memórias TCAM, que trabalham em altíssimas velocidades, são usadas para a implementação de funcionalidades como *firewalls*, NAT, QoS e coleta de estatísticas. O *OpenFlow* estabelece um protocolo para o acesso e a manipulação dessas tabelas de fluxo. Dessa forma é possível controlar como cada equipamento da rede tratará cada fluxo que passa por ele, atingindo o objetivo inicial de isolamento de fluxos distintos. Entretanto, esse protocolo também permite que esse controle seja feito programaticamente, o que abre espaço para muitas outras possibilidades.

2.3.1 Elementos de uma rede *OpenFlow*

Uma rede baseada em *OpenFlow* é composta por um ou mais equipamentos que suportam essa tecnologia e por um controlador. Um equipamento que suporta *OpenFlow* é chamado de *switch OpenFlow*. Um *switch OpenFlow* pode ser um equipamento fabricado conforme as especificações do protocolo *OpenFlow* (OPEN NETWORKING FOUNDATION, 2011). Trata-se de um equipamento convencional adaptado, ou até um elemento de *software* que implemente as

especificações do *OpenFlow*, como um *switch* virtual (PFAFF, 2009). Seguindo a definição de McKeown et al., um *switch OpenFlow* apresenta as seguintes características: suporta o protocolo de comunicação, possui uma tabela de fluxos, que é manipulada através do protocolo e implementa um canal de comunicação seguro para se conectar ao controlador. O controlador é um *software* que executa em um computador (convencional) e que se conecta a todos os *switches OpenFlow* da rede e manipula suas tabelas de fluxos. Com base nas entradas de suas tabelas de fluxos, os *switches OpenFlow* realizam o encaminhamento dos dados através da rede. Todo o controle da rede é centralizado e por isso sua programação é facilitada. Dessa forma, o *OpenFlow* implementa o conceito de RDS de separação entre planos, com a vantagem de que o plano de dados é implementado pelos *switches*, que trabalham em velocidades de *hardware*, reduzindo o impacto da separação no desempenho da rede. A Figura 2.3 ilustra o funcionamento de uma rede *OpenFlow*.

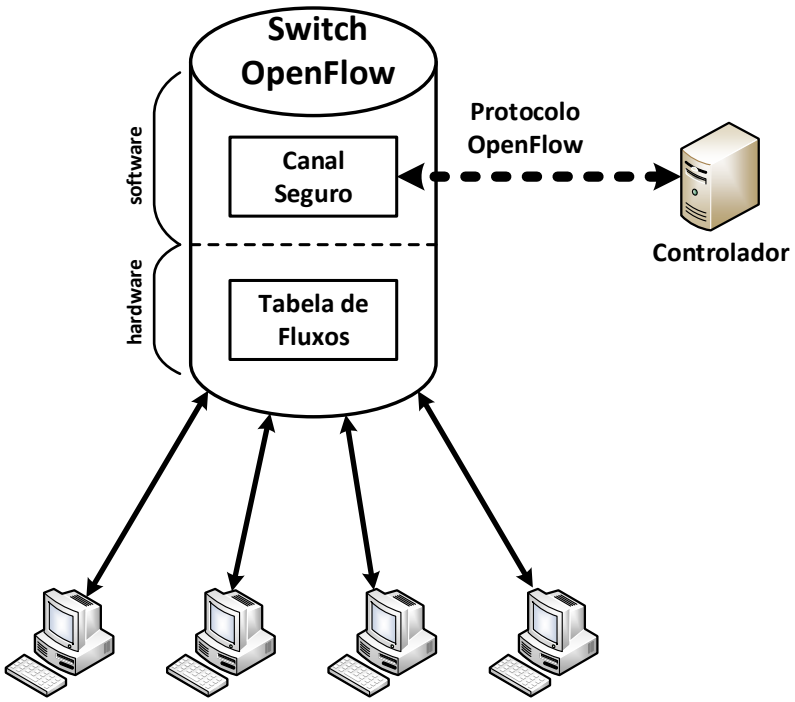


Figura 2.3: Funcionamento de uma rede *OpenFlow*. Adaptado de McKeown et al.

2.3.2 Tabela de fluxos

Em uma rede *OpenFlow*, os *switches* não tomam nenhum tipo de decisão de roteamento ou encaminhamento. Todas as decisões são tomadas pelo controlador e repassadas a eles. Isso é feito através da manipulação de entradas nas tabelas de fluxo. Um fluxo é caracterizado como

é uma sequência de pacotes que possuem características comuns. *OpenFlow* utiliza campos de cabeçalho de pacotes para a classificação dos mesmos em fluxos. *OpenFlow* utiliza valores de 12 campos de cabeçalho para caracterizar um fluxo[2], exibidos na Figura 2.4. Um fluxo pode ser identificados pela correspondência exata (”*match”*) dos valores de todos os 12 campos ou pela correspondência de um subconjunto dessa 12-upla. No segundo caso são utilizados caracteres especiais (denominados *wildcards*) para o preenchimento das entradas na tabela de fluxos do *switch*. Essa estrutura de campos permite alta flexibilidade no encaminhamento de pacotes, pois um fluxo pode ser encaminhado não apenas baseado no endereço IP de destino, como ocorre nas redes TCP/IP convencionais, mas também baseado em portas TCP (ou UDP), em endereços MAC, etc. Os elementos de encaminhamento do *OpenFlow* são chamados de *switches* pois os fluxos podem ser atribuídos com base em endereços da camada 2. Essa nomenclatura, no entanto, não implica que o encaminhamento deva necessariamente ocorrer baseado na camada 2.

Porta ingresso	MAC orig	MAC dest	Tipo Ether	Id VLAN	prioridade VLAN	IP orig	IP dest	IP proto	IP TOS	Porta orig TCP/UDP	Porta dest TCP/UDP

Figura 2.4: Estrutura de campos de cabeçalho que caracteriza um fluxo *OpenFlow*. Adaptado de McKeown et al.

Uma entrada na tabela de fluxos de um *switch OpenFlow*, ilustrada na Figura 2.5 é composta de três partes: uma estrutura que identifica os fluxos, um conjunto de contadores e um conjunto de uma ou mais ações. Os contadores são utilizados para computação de dados estatísticos, como o número de pacotes e de *bytes* processados por aquele fluxo, erros e o instante de tempo (*timestamp*) em que o último pacote desse fluxo foi processado pelo *switch*. Essa informação é utilizada para auxiliar na remoção de entradas da tabela correspondentes a fluxos inativos. A Seção 2.3.3 explica em detalhes esse processo. As ações definem como os pacotes de cada fluxo serão processados. A especificação do protocolo *OpenFlow* define três ações básicas que todos os equipamentos devem suportar:

1. Encaminhar os pacotes do fluxo para uma ou mais portas do *switch*;
2. Encapsular os pacotes do fluxo em uma mensagem do protocolo de comunicação e enviar para o controlador;

[2]Os conceitos apresentados neste trabalho são baseados na versão 1.0 do protocolo *OpenFlow*, pois é a versão utilizada na implementação do mesmo, e a versão implementada pela maioria dos equipamentos que o suportam. A versão mais recente do protocolo é a 1.4, e sua especificação pode ser encontrada em https://www.opennetworking.org/sdn-resources/onf-specifications/openflow

3. Descartar os pacotes do fluxo;

Além dessas, a especificação define outras ações, que caracteriza como opcionais, para expandir a funcionalidade dos *switches OpenFlow*. Uma dessas ações, apesar de opcional, é implementada por praticamente todos os equipamentos que suportam *OpenFlow* atualmente, por sua grande utilidade:

4. Modificar valores em determinados campos de cabeçalho dos pacotes do fluxo.

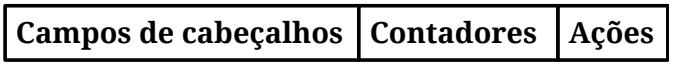


Figura 2.5: Entrada de uma tabela de fluxos. Adaptado de McKeown et al.

2.3.3 Funcionamento de uma rede *OpenFlow*

Em uma rede *OpenFlow* todas as decisões são centralizadas no controlador. Conforme cresce o tamanho da rede gerenciada por um controlador, ou aumenta o tráfego na rede, ele poderia tornar-se um gargalo, já que todos os *switches* da rede o consultam. Entretanto, o fato dos *switches* serem orientados a fluxos diminui esse problema, tornando as redes *OpenFlow* mais escaláveis. O gerenciamento dos fluxos em uma rede *OpenFlow* é descrito a seguir.

Quando um pacote chega em uma das interfaces de um dos *switches* da rede, ele compara os valores dos seus campos de cabeçalhos com os valores das entradas de sua tabela de fluxos. Caso o pacote não case com nenhuma das entradas da tabela, ou seja, ele não pertence a nenhum dos fluxos conhecidos por ela, ele é encapsulado em uma mensagem do protocolo de comunicação e enviado para o controlador através do canal seguro. O controlador recebe o pacote, o analisa baseado na lógica de controle que ele implementa e decide qual, ou quais, ações devem ser tomadas pelo *switch* para esse pacote e para todos os subsequentes a ele, ou seja, ao fluxo. O controlador então envia essas informações ao *switch*, que as converte em uma nova entrada em sua tabela de fluxos. A partir desse momento, todos os pacotes desse novo fluxo que chegarem até o *switch* serão identificados e tratados conforme as ações da entrada da tabela, não sendo necessário se comunicar com o controlador. Dessa forma, o controlador consegue controlar diversos *switches* sem se tornar um gargalo.

As tabelas de fluxos possuem um número finito de entradas. Para evitar que elas sejam completamente preenchidas e entradas sejam descartadas, comprometendo a funcionalidade

da rede, há um mecanismo baseado em tempos de expiração (*timeouts*). Cada entrada da tabela de fluxos possui associada a si dois limitadores de tempo, denominados *hard_timeout* e *idle_timeout*. O primeiro determina o tempo máximo que cada entrada pode permanecer na tabela, e após esse tempo ela é removida. O segundo é acionado para remover entradas inativas na tabela. Se nenhum pacote for processado por aquela regra durante esse período de tempo, a regra também será removida. Caso a regra seja removida antes da comunicação entre os nós ser encerrada, o próximo pacote a chegar em um dos *switches* da rede será tratado como o primeiro pacote do fluxo. É importante ressaltar que o fluxo de pacotes dentro da rede *OpenFlow* não necessariamente é o mesmo fluxo das aplicações que executam sobre ela.

2.3.4 Sistema operacional de rede

Como visto nas seções anteriores, a especificação do protocolo *OpenFlow* define a comunicação entre o controlador e os *switches*. Entretanto, toda a lógica da rede cabe ao controlador. Como existem diversas funcionalidades desejáveis para as redes, surge a necessidade de mecanismos para facilitar a criação de novas funcionalidades. Gude et al. (GUDE, 2008) introduz o conceito de sistema operacional de rede (*Network Operating System* - NOS). A ideia central diz que assim como sistemas operacionais modernos criam abstrações para os detalhes de baixo nível do computador, facilitando o desenvolvimento de novas aplicações, também o *OpenFlow*, com sua comunicação padronizada e visão centralizada e global da rede, abstrai os detalhes de baixo nível da rede. Em seu trabalho ele apresenta o NOX, que utiliza essa visão e cria sobre ela uma interface programática (API) orientada a eventos globais da rede, para facilitar a criação de novas aplicações controladoras. Ele também permite a execução de múltiplas aplicações simultaneamente, de forma análoga aos sistemas operacionais.

2.4 Migração e continuidade de serviço

Tem-se visto um grande número de companhias tornando-se provedores de IaaS, seja para vender serviços (para o público em geral), ou apenas para prover recursos para seus próprios negócios. Consequentemente, vários *datacenters* provendo serviços de nuvem vem sendo instalados em diversos lugares. A abundância dos recursos de *hardware* traz à tona o interesse de interoperar entre eles, e de migrar aplicações entre nuvens. Proprietários de nuvens privadas podem querer migrar suas aplicações para uma nuvem pública para atender a picos de demanda (*cloudbursting*). Clientes de nuvens públicas podem adquirir sua infraestrutura privada e quererem migrar suas aplicações para lá. Instituições proprietárias de nuvens públicas podem querer

federar seus recursos, de modo a se beneficiarem mutuamente. Clientes de provedores públicos podem querer migrar para outro provedor, buscando melhores preços, ou melhores garantias de QoS. Além disso, há outros motivos em que a migração se torna interessante, como migrar uma aplicação para mais perto de um grupo de usuários, para melhorar a qualidade de serviço, ou, em casos de picos de demanda, migrar aplicações para locais onde os custos com energia elétrica e/ou refrigeração sejam menores, o que é conhecido como *"follow the sun and follow the wind"* (BOUGHZALA, 2011), ou até por motivos relativos a segurança da informação.

Apesar disso, a migração entre diferentes *datacenters* apresenta uma grande dificuldade: manter a continuidade de serviço. Ao se transferir uma aplicação para um novo provedor, ela passa a executar em uma máquina diferente, com um endereço IP diferente, interrompendo todas as comunicações até que os clientes sejam notificados desse novo IP. Em tempos em que a interrupção de serviços é cada vez mais indesejável, e pode gerar grandes prejuízos, manter o serviço responsivo durante uma migração entre provedores ganha fundamental importância para permitir a interoperabilidade entre diferentes provedores. Sendo assim, no presente trabalho procurou-se descobrir quais são as técnicas utilizadas atualmente para resolver esse tipo de problema.

Dentro dos limites do *datacenter*, a migração sem interrupção de serviço é um problema praticamente resolvido. A partir do trabalho de Clark et al. (CLARK, 2005), é introduzido o conceito de migração ao vivo (*live migration*) de máquinas virtuais, onde os autores transferem uma MV completa de uma máquina física para outra sem interromper sua execução. Tendo como pré-requisitos as máquinas possuírem o mesmo hipervisor, estarem na mesma rede local (LAN), e compartilharem os dados referentes ao disco da MV a ser migrada, os autores utilizam o hipervisor para realizar a migração basicamente transferindo dados da memória e do estado do processador de uma maneira otimizada, e utilizam o protocolo ARP para garantir que a MV mantenha seu endereço IP na nova máquina física. Dessa forma eles conseguem garantir a continuidade da execução da MV durante todo o processo. Fora dos limites do *datacenter*, entretanto, uma migração desse tipo é mais complexa. Na literatura, a maioria dos trabalhos relacionados à migração sem interrupção de serviço tratam da migração de MVs em diferentes contextos.

Hirofuchi et al. (HIROFUCHI, 2009) apresentam um mecanismo para a migração ao vivo de MVs através de redes de longa distância, num contexto de *clusteres* virtuais. O objetivo principal do trabalho consiste em otimizar a transferência de dados de disco de MVs entre redes de longa distância, a fim de permitir o processo de migração. Entretanto, no trabalho são discutidas soluções para a migração de dados de memória e estado de MVs em redes de

longa distância e manutenção do acesso à MVs durante e após operações de migração. Os autores citam outro trabalho (HIROFUCHI, 2008), que define melhor o ambiente onde ocorrem as migrações e mostram uma forma de garantir essa consistência.

A princípio é definido um ambiente de *cluster* virtual. Trata-se de uma infraestrutura de *cluster*, onde todos os nós possuem hipervisores e instanciam MVs. Dessa forma, o *cluster* físico é dividido em múltiplos ambientes de *cluster* virtuais. O ambiente possui uma interface centralizada, que permite instanciá-los rapidamente, e configurar S.O. e programas a serem instalados em todas as MVs de uma vez. Cada ambiente virtual é interconectado através de uma rede local privada, e as redes privadas são isoladas entre si através de VLANs. A seguir é definido um *cluster* virtual multi-sítio, que é um ambiente onde outras infraestruturas como essa, localizadas em sítios separados, são interconectadas através de redes de longa distância, e os *clusters* virtuais podem ser instanciados utilizando recursos dos diversos sítios.

Por fim, são explicadas as ideias e tecnologias usadas para isso. Para que as MVs de um *cluster* virtual distribuído entre diferentes sítios se comuniquem como em uma rede local, os autores desenvolveram um mecanismo para estender a rede local privada entre os sítios. Para isso, eles adicionam um novo componente em cada sítio, para interconectá-los através de VPNs. A Figura 2.6 ilustra esse mecanismo: quando um *cluster* virtual é instanciado em mais de um sítio, são instanciados também um agente de VPN em cada um referente ao novo ambiente, que também é conectado ao segmento de VLAN. Quando as MVs se comunicam com os recursos de outros sítios, os dados são encaminhados para o agente de VPN, que as encaminha para o outro sítio. Dessa forma, é mantido o mesmo ambiente de rede local entre todo o *cluster* virtual. Além disso, são preservados os identificadores de VLAN: antes de enviar para o outro sítio, os dados da VLAN são removidos dos pacotes. Dessa forma não é preciso atribuir o mesmo identificador de VLAN entre sítios.

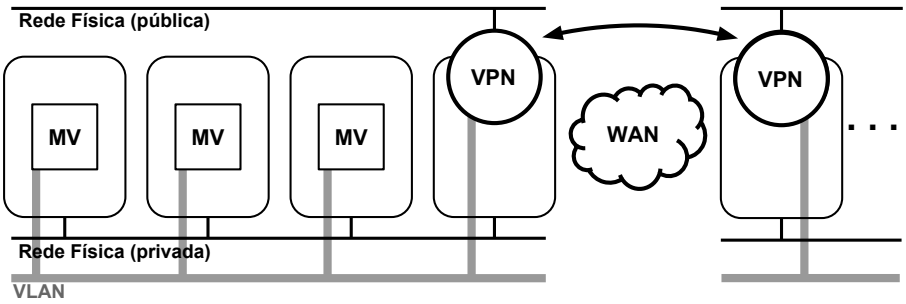


Figura 2.6: Extensão de rede local com VPNs. Adaptado de Hirofuchi et al.

Os trabalhos de Nagin et al. (NAGIN, 2011) e de Hadas et al. (HADAS; GUENENDER; ROCHWERGER, 2009) tratam de nuvens federadas. Segundo os autores, nuvens federadas são

ambientes onde diferentes provedores de computação em nuvem, ou, mais especificamente, de IaaS, através de acordos de benefício mútuo, agregam seus recursos computacionais em um grande sistema distribuído, permitindo aos seus clientes utilizarem recursos de qualquer um dos provedores federados de maneira transparente e independente de localidade. Em nuvens federadas, cada provedor participante possui seus próprios objetivos de negócio, e provavelmente são concorrentes entre si. Dessa forma, o grande desafio nesse caso é agregar os recursos de infraestrutura, porém mantendo a autonomia, a privacidade e a segurança de cada provedor, o que eles denominam princípios de insularidade.

Dentro desse contexto, Nagin et al. propõem um sistema para realizar a migração ao vivo e transparente de MVs entre diferentes provedores pertencentes à mesma federação de nuvens. O sistema é composto de três componentes que migram dados de disco, memória e rede virtual, além de implementar um elaborado sistema de controle de acesso, para garantir os requisitos de privacidade entre os provedores federados. A migração de redes virtuais é feita de tal maneira que as MVs em migração mantenham seu endereço IP durante e após o processo. Para tanto, é utilizado o trabalho de Hadas et al., que implementa um serviço de redes virtuais.

O serviço de criação de redes virtuais (chamadas pelos autores de *Virtual Application Networks*, ou VANs) proposto por Hadas et al. procura criar um mecanismo em que clientes possam não apenas instanciar MVs em qualquer nuvem da federação, mas também configurar uma rede local entre elas dinamicamente, independente da localização das mesmas. Além disso, busca garantir os princípios de insularidade das nuvens federadas, tornando o tráfego independente da localização das MVs. O serviço resultante funciona como exibido na Figura 2.7, onde várias VANs podem ser instanciadas e as MVs anexadas a elas dinamicamente. MVs podem fazer parte de mais de uma VAN, possuindo uma interface de rede conectada a cada uma, e as VANs podem conectar MVs de um ou mais sítios, todas em uma mesma rede virtual local, isolada das outras.

As VANs foram implementadas como uma rede de sobreposição (*overlay network*). Em cada máquina física dos *datacenters* são criadas interfaces virtuais que implementam essa rede quando se deseja instanciar uma nova VAN ou anexar uma MV a uma VAN existente. Essa interface encapsula os quadros Ethernet criados pelas MVs em quadros de VAN, que contêm informações que são usadas não só para identificar a qual VAN cada quadro pertence, mas também para permitir o encaminhamento fim-a-fim dos pacotes, ao invés do encaminhamento nó-a-nó padrão do protocolo Ethernet. Os quadros VAN, por sua vez, são encapsulados em datagramas UDP e então enviados aos nós físicos da rede pertencentes à VAN correspondente. As Figuras 2.8(a) e 2.8(b) mostram o esquema de encapsulamento dos pacotes e o cabeçalho

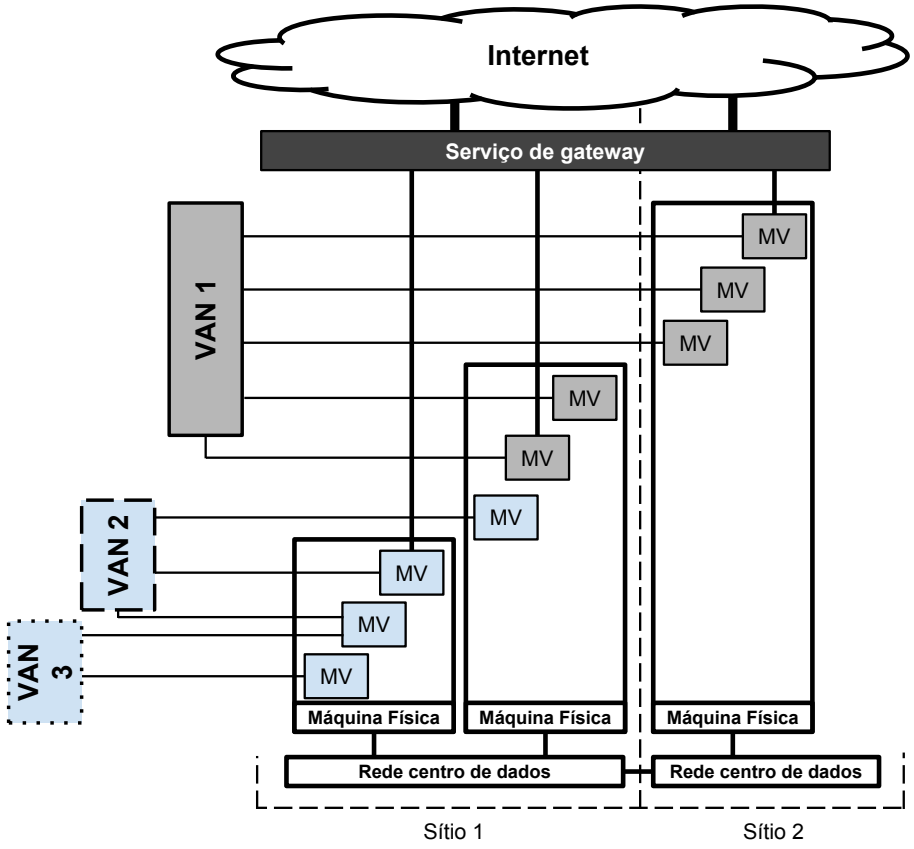
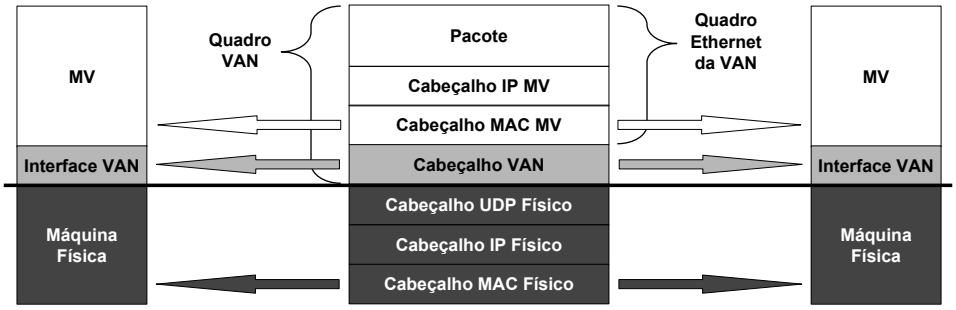


Figura 2.7: Exemplos de VANs em um ou mais sítios. Adaptado de Hadas et al.

padrão dos quadros VAN. Para que seja possível manter MVs de sítios diferentes na mesma VAN, uma interface em um nó de cada centro de dados é configurada como *proxy*. Todos os pacotes endereçados a nós de VANs que estejam em outro sítio são encaminhados ao *proxy*, que estabelece conexões seguras com outro *proxy* localizado no sítio destino, e que trabalha de maneira análoga.

Bradford et al (BRADFORD,) apresenta um sistema para a migração ao vivo de MVs entre redes de longa distância (*Wide Area Networks*, ou WANs). Esse trabalho tem como objetivo utilizar uma combinação de técnicas e ferramentas bem conhecidas para realizar essa migração. O sistema apresentado foi construído como uma extensão do XenoServer (HAND,) uma plataforma distribuída para gerenciamento e instanciação de MVs entre diferentes sítios de diferentes organizações, que por sua vez, utiliza como base o hipervisor Xen (BARHAM, 2003). O sistema realiza a migração de dados de disco utilizando um *driver* customizado para o Xen, e do estado das MVs utilizando a funcionalidade de migração ao vivo do Xen. A manutenção da conectividade durante e após a migração é implementada através de uma combinação entre tunelamento IP sobre IP e DNS dinâmico.

Nesse sistema, as MVs possuem endereços IP públicos visíveis através da Internet e são



(a) Encapsulamento Ethernet sobre VAN sobre UDP

Versão	Tipo	Flags	ID VAN Origem
ID VAN Destino			
Pacote MAC da VAN			

(b) Cabeçalho VAN

Figura 2.8: Detalhes do protocolo VAN. Adaptado de Hadas et al.

acessadas pelos clientes através de seus nomes de domínio, resolvidos por um servidor de DNS dinâmico. No processo de migração são transferidos os dados de disco enquanto a MV continua a atender clientes; a consistência de disco entre origem e destino é mantida por um *driver* customizado. Ao fim da transferência dos dados de disco, a MV é suspensa na origem e seus dados de memória e estado são transferidos e ela é retomada no destino, porém com duas interfaces de rede: uma com seu IP e outra com o IP da MV na origem. A máquina física na origem cria um túnel IP sobre IP (utilizando a ferramenta *iproute2*) e começa a redirecionar todo o tráfego de rede destinado à MV para o destino. A máquina destino também cria um túnel para encaminhar as respostas das requisições à máquina física na origem, que as encaminha aos clientes. Dessa forma, as conexões em andamento no momento da migração não são interrompidas. Quando a MV é retomada no destino, sua entrada de nome de domínio é atualizada no DNS dinâmico, fazendo com que novos clientes façam suas requisições diretamente em seu novo IP. O túnel continua aberto até que se encerrem as conexões em andamento. Após isso ele é fechado e a MV passa a responder apenas pelo IP do destino.

Em 2005, Travostino et al. (TRAVOSTINO, 2006) apresentaram um sistema para a migração de MVs entre diferentes sítios separados por grandes distâncias. O sistema apresentado foi desenvolvido para ambientes em que diferentes organizações agregam recursos de seus *clusteres* computacionais, que são configurados utilizando um *middleware* comum. Esses ambientes geralmente são heterogêneos e, por serem multi-institucionais, há uma grande preocupação com autorização e autenticação, além de uma série de políticas complexas para uso dos recursos. O sistema apresentado é formado por um agente central que coordena todo o processo, por agentes

em cada sítio do ambiente, que realizam diferentes operações, como autenticação e autorização, reserva de enlaces de rede entre a origem e o destino e transferência de dados de disco entre sítios. Além desses, há os hipervisores executando nas máquinas de cada sítio, que são responsáveis pela migração do estado das MVs em migração. A manutenção da conectividade dos clientes nesse sistema é feita através de túneis IP e de configurações no hipervisor das máquinas de origem e de destino, como ilustrado no exemplo da Figura 2.9.

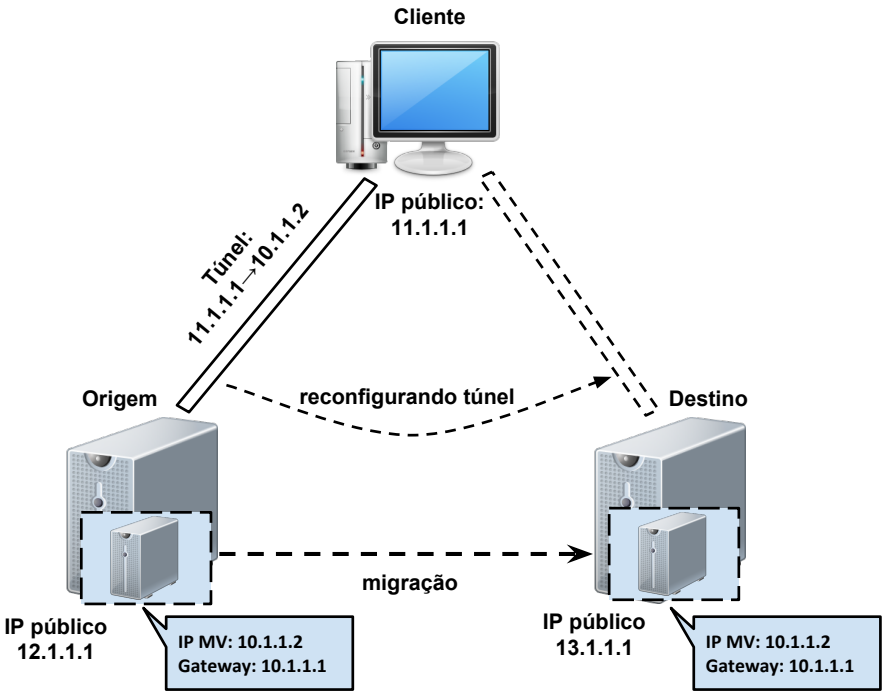


Figura 2.9: Esquema de tunelamento IP para migração de máquinas virtuais, adaptado de (TRAVOSTINO, 2006)

A MV possui um endereço privado(no exemplo, 10.1.1.2), e seu acesso à rede externa ocorre por meio de um *gateway* virtual, criado pelo hipervisor (no exemplo, 10.1.1.1)[3]. Os clientes acessam as aplicações executando na MV através de um túnel IP, de forma que os pacotes são endereçados diretamente ao endereço do *gateway*, e são tunelados através do endereço público (no exemplo, 12.1.1.1). Quando ocorre a migração, a MV é instanciada no destino com o mesmo endereço IP privado e *gateway* virtual. O agente global responsável por coordenar o processo reconfigura o túnel IP dinamicamente com o endereço público do servidor no sítio de destino da MV (no exemplo, 13.1.1.1). Dessa forma, a migração é transparente para o cliente e as conexões TCP não são quebradas durante e após a migração.

[3]O gateway é criado para que outras MVs possam ser instanciadas e migradas.

2.4.1 Comparação entre os trabalhos

Analisando os trabalhos apresentados lado a lado, como exibido pelo quadro comparativo da Tabela 2.1, pode-se notar alguns pontos. O primeiro deles é que todas as tecnologias e estratégias empregadas pelos autores para manter a conectividade entre MVs através de WANs eram baseadas no encapsulamento de pacotes. Apesar de ser uma técnica comum atualmente, ela apresenta algumas desvantagens, como o *overhead* de encapsulamento na origem e desencapsulamento no destino, e o aumento na taxa de fragmentação de pacotes, visto que o encapsulamento aumenta o tamanho dos pacotes transmitidos. Outro ponto notório, comum a todos os trabalhos é a necessidade de controle de recursos da infraestrutura dos *datacenters* para a implementação dos mecanismos de manutenção de conectividade. Em geral, são utilizados recursos de hipervisores, ou dependem da configuração dos recursos do *datacenter*. Além disso, nota-se uma diferença importante no trabalho de Bradford et al. em relação aos outros trabalhos analisados: ele não procura manter o endereço IP após a migração, mas sim utiliza um DNS dinâmico para notificar os usuários da MV migrada ao seu novo endereço.

Trabalho	Cenário	Mantém IP Após migração?	Tecnologia	Encapsulamento?	Controle da infraestrutura?
Hirofuchi et al.	Cluster virtualizado multi-sítio	Sim	VPN	IPSec sobre IP	Sim
Nagin et al.	Nuvens Federadas	Sim	*Overlay Network*	Ethernet sobre VAN sobre UDP	Sim
Bradford et al.	Grade / Virtualização	Não	Tunelamento IP + DNS dinâmico	IP sobre IP	Sim
Travostino et al.	Grade / Virtualização	Sim	Tunelamento IP dinâmico	IP sobre IP	Sim

Tabela 2.1: Quadro comparativo dos trabalhos sobre migração e continuidade de serviço.

Dessa forma, as soluções atuais para a manutenção da conectividade e da continuidade de serviço durante a migração em WANs ainda apresentam pontos que podem ser melhorados. Com base nas características do *OpenFlow*, especificamente sua capacidade de manipular fluxos de rede através de operações de reescrita de campos de cabeçalho executadas por equipamentos de rede em altas velocidades, e devido a sua natureza aberta, começou a se formular a hipótese de que que houvesse espaço para utilizar *OpenFlow* nesse contexto. Sendo assim, procurou-se na literatura trabalhos que utilizam *OpenFlow* para redirecionamento de fluxos, para obter uma visão do que poderia ser feito utilizando essa tecnologia, e se seria possível empregá-la no cenário de migração entre provedores de nuvem IaaS. Alguns dos trabalhos analisados são descritos na Seção seguinte.

2.5 Redirecionamento de fluxos com *OpenFlow*

O trabalho de Boughzala et al. (BOUGHZALA, 2011) propõe o uso de *OpenFlow* dentro de redes de centros de dados para aumentar sua flexibilidade e dar suporte à interconexão de ambientes para diferentes fins, tais como a migração de aplicações entre domínios, visando aproximá-la dos clientes e melhorar a experiência do usuário. Os autores discutem as dificuldades encontradas em interconectar redes de centros de dados diferentes devido a suas diferentes topologias e suas incompatibilidades. Ao estudar com detalhes algumas dessas topologias eles mostram, entretanto, que apesar de incompatíveis, essas topologias apresentam algumas características em comum.Com base nessas características e na facilidade de se configurar uma rede baseada em *OpenFlow*, devido à centralização de seu controle, os autores apresentam uma solução para facilitar a interconexão entre diferentes centros de dados e outras operações.

A solução proposta é um *middleware* distribuído capaz de reconfigurar dinamicamente redes de centros de dados baseadas em *OpenFlow*. A ideia central é abstrair as características de conectividade das redes de cada centro de dados em regras de alto nível, independente de topologias de rede, de protocolos, etc. Com base nessa especificação de alto nível, o *middleware* a traduz para regras *OpenFlow* que são enviadas para os centros de dados e instaladas em todos os equipamentos de suas redes, permitindo assim sua interconexão. Os testes realizados pelos autores em ambientes simulados mostraram que a configuração dinâmica de redes *OpenFlow* baseada no mecanismo proposto consegue estabelecer a conectividade das redes rapidamente, o que os motiva a dizer que com essa ideia é possível, inclusive, criar um serviço de configuração de rede sob demanda, baseado nos conceitos de IaaS.

O trabalho de Hao et al. (HAO, 2010) utiliza conceitos de RDS no contexto de computação em nuvem e mobilidade de aplicações, visando aproximar a aplicação do usuário. Os autores propõem uma arquitetura de rede para permitir a migração de MVs entre diferentes redes mantendo seu endereço IP.

A ideia central do trabalho é combinar equipamentos rede (como *switches* e roteadores) geograficamente distribuídos em um único roteador lógico, ou nas palavras dos autores um roteador aberto virtualmente agregado (*Virtually Clustered Open Router, ou VICTOR*). A arquitetura desse roteador virtual é exibida na Figura 2.10. Os elementos encaminhadores de pacotes (EEs), distribuídos entre diferentes redes da camada três, são todos interconectados, seja por ligações físicas dedicadas, circuitos virtuais estabelecidos por MPLS, ou túneis IP. Estes EEs são gerenciados por um controlador centralizado (CC), encarregado de definir o encaminhamento de pacotes de todo o conjunto, realizando as computações necessárias e distribuindo as

tabelas de encaminhamento.

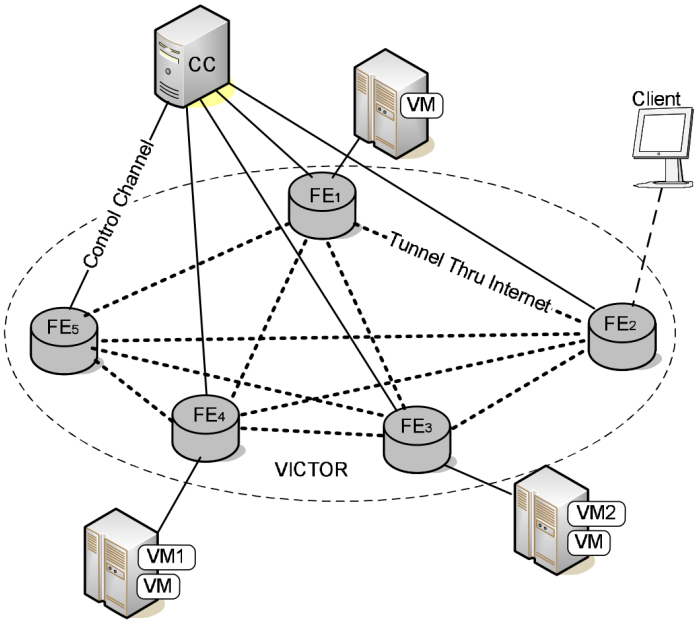


Figura 2.10: Arquitetura da plataforma VICTOR. Adaptado de Hao et al.

O CC mantém uma tabela com a topologia dos EEs, tendo conhecimento de quem está conectado a quem. Ele também mantém um registro de todas as MVs que estão conectadas aos EEs. Esse registro é feito através dos prefixos dos seus endereços IP, de forma que cada MV é associada a apenas um EE em um determinado instante. Quando uma nova MV se conecta a um EE, ela é registrada pelo CC; quando alguma MV é migrada ou desconectada, suas informações são atualizadas ou removidas do CC. Com base nessa visão global da rede, ele realiza os cálculos e distribui as tabelas de roteamento entre todos os elementos da rede, mantendo assim a conectividade durante todo o tempo.

Neste trabalho os autores conseguem prover a continuidade de serviço mantendo o controlador sempre atualizado de toda a topologia da rede, e de todas as máquinas que podem ser migradas, de modo que ele pode alterar os fluxos de rede livremente dentro de sua rede, mantendo assim a conectividade das MVs mesmo quando elas mudam de rede. Entretanto, manter todo o estado acaba fazendo com que essas tabelas cresçam muito rápido, conforme aumenta o número de nós na rede, e isso pode acarretar em problemas para atualizar os EEs, visto que as tabelas de fluxo dos equipamentos têm tamanho limitado.

O trabalho de Nunes et al. (NUNES; PONTES; GUEDES,) utiliza *OpenFlow* buscando otimizar o gerenciamento de redes virtuais dentro de centros de dados de provedores IaaS. Tipicamente, esses provedores possuem múltiplos clientes, que podem instanciar várias máquinas virtuais, e criar redes virtuais para interconectá-las. Dessa forma, é um requisito essencial que redes virtu-

ais de diferentes clientes sejam isoladas umas das outras, mesmo que todas elas compartilhem a mesma infraestrutura física. Os autores propõem uma abordagem baseada em *OpenFlow* para garantir esse isolamento, em oposição aos métodos utilizados até então, baseados em empilhamento de protocolos (tunelamento) ou roteamento da camada 3.

A solução é descrita pelos autores com base no protótipo funcional desenvolvido, e nas tecnologias utilizadas para isso. Ela consiste de uma aplicação controladora chamada *DCPortals*, desenvolvida sobre o POX, que interage com o gerenciador de nuvens *OpenStack*. *OpenStack* gerencia uma infraestrutura física para a instanciação de máquinas virtuais (MVs) sob demanda. As máquinas virtuais são gerenciadas pelo VMM Xen. As redes virtuais entre as máquinas são implementadas através de *switches* virtuais *openvswitch* instalados em cada nó físico com Xen. O sistema funciona da seguinte forma: quando um cliente solicita a instanciação de máquinas virtuais, o gerenciador *OpenStack* armazena informações sobre essas máquinas em uma base de dados interna e a seguir instrui o monitor de máquinas virtuais Xen a instanciá-las nos nós físicos. *DCPortals* acessa a base de dados do *OpenStack* para determinar um identificador único de rede virtual para esse conjunto de MVs e usa essas informações para controlar os *switches* virtuais executando nas máquinas físicas do centro de dados.*DCPortals* se conecta a todos os *switches* virtuais de todo o centro de dados, e obtém informações sobre todas as MVs que estão instanciadas. Dessa forma ele tem uma visão global tanto da rede física quanto das redes virtuais, e com essas informações, consegue isolar os tráfegos, impedindo que tráfego de uma rede virtual alcance MVs de outras redes virtuais. O mesmo tratamento é dado para pacotes *broadcast*, que são propagados apenas entre as MVs da mesma rede virtual. Dessa forma o volume de dados em trânsito na rede do *datacenter* é reduzida.

Para otimizar o isolamento dos tráfegos é utilizada a reescrita de campos de cabeçalho, ilustrada na Figura 2.11. De posse das informações que coleta dos *switches* virtuais e da base de dados do *OpenFlow*, *DCPortals* consegue saber em qual máquina física se encontra cada uma das MVs em execução. Quando MVs que fazem parte da mesma rede virtual, mas estão localizadas em máquinas físicas diferentes, desejam se comunicar, *DCPortals* instrui os *switches* virtuais a reescreverem os pacotes antes de os encaminharem para a rede física com o endereço da máquina física que abriga a MV destino do pacotes. Quando os pacotes chegam na máquina física de destino, o *switch* virtual nela é instruído, então, a reescrever novamente os pacotes com o endereço da MV. Dessa forma, pela rede física do centro de dados trafegam apenas pacotes relacionados às máquinas físicas. Dessa forma evita-se que as tabelas de encaminhamento dos *switches* físicos da rede sejam sobrecarregadas sem a necessidade de encapsulamento de pacotes. A natureza centralizada e orientada a fluxos do *OpenFlow* garante que o desempenho das operações necessárias ao isolamento das redes virtuais não seja alterado conforme a quantidade

de MVs e redes virtuais no centro de dados aumenta, ao contrário de um isolamento baseado em roteamento da camada 3.

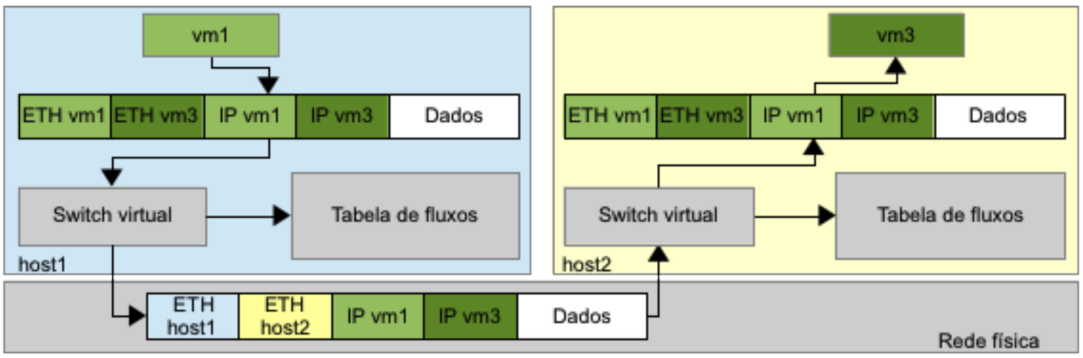


Figura 2.11: Operações de reescrita de cabeçalhos do *DCPortals*. Adaptado de Nunes et al.

2.5.1 Pontos a destacar

A análise dos trabalhos que usam *OpenFlow* revelou alguns pontos importantes que se relacionam diretamente com o tema deste trabalho. Os trabalhos de Boughzala et al. e de Hao et al. indicam que o *OpenFlow* oferece recursos para a manutenção da conectividade em operações de migração, seja dentro da rede do *datacenter*, entre redes de *datacenters* diferentes, ou em sítios distribuídos geograficamente, como mostra o trabalho de Hao et al. Já o trabalho de Nunes et al., apesar de não tratar de operações de migração, apresenta dois resultados interessantes. O primeiro deles é mostrar as vantagens da utilização de *OpenFlow* na alteração de fluxos de rede em relação ao tunelamento, que é baseado no encapsulamento de pacotes. O trabalho também mostra como a utilização de *OpenFlow* em combinação com *switches* virtuais pode agregar a flexibilidade e as funcionalidades de RDS a redes sem a necessidade da utilização de hardware específico.

2.6 Conclusões

Com base nas análises feitas dos trabalhos relacionados, pode-se chegar às seguintes conclusões:

- A manutenção da continuidade de serviço em migrações entre WANs apresenta oportunidades para a contribuição de pesquisa, visto que ainda apresenta soluções limitadas, baseadas principalmente em técnicas de encapsulamento de pacotes;

- *OpenFlow* já tem sido usado procurando melhorar as atuais técnicas dentro de redes de grandes *datacenters*, e também em WANs;
- Resultados de trabalhos anteriores mostram as vantagens do *OpenFlow* em relação às atuais técnicas utilizadas para esse fim;
- Dadas as características do *OpenFlow*, ele parece se encaixar no cenário de migração entre provedores de nuvem IaaS, mesmo em cenários com acesso limitado à recursos de infraestrutura, com o uso de *switches OpenFlow* virtuais;

Concluídos esses pontos, o Capítulo 3 apresenta a solução baseada em *OpenFlow* proposta neste trabalho.

Capítulo 3

PROPOSTA DE ARQUITETURA PARA REDIRECIONAMENTO DE FLUXOS

Este capítulo apresenta a proposta de uma arquitetura para redirecionamento de fluxos de rede, cujo objetivo é dar suporte à migração de aplicações entre *datacenters* de provedores de infraestrutura como serviço. Também são descritos cenários de utilização desta arquitetura, seu funcionamento no contexto de uma migração de aplicação, bem como os principais aspectos do projeto e da implementação do protótipo funcional.

3.1 Projeto da arquitetura

Migrar aplicações entre *datacenters* de provedores de IaaS é um recurso interessante em uma série de cenários diferentes: seja entre diferentes sítios de um mesmo provedor (público ou privado), de uma nuvem privada para um provedor público, entre dois provedores públicos, entre outros casos. Ao se projetar uma arquitetura para dar suporte à migração de aplicações é desejável que esta seja aplicável à maioria desses cenários. Sendo assim, observam-se os seguintes pontos:

- A maioria dos provedores de IaaS utiliza virtualização em seus *datacenters*, oferecendo seus recursos na forma de máquinas virtuais;
- A migração entre dois *datacenters* virtualizados, ambos utilizando o mesmo hipervisor, é uma operação atualmente suportada pelos fabricantes de hipervisores. Em geral, provedores que possuem vários *datacenters* utilizam o mesmo hipervisor em todos, o que facilita essa operação;

- Entre *datacenters* virtualizados de dois provedores privados distintos pode não haver a mesma uniformidade de hipervisores ou compatibilidade de tecnologias, o que pode dificultar processos de migração. Entretanto, cada um dos provedores tem total controle sobre sua infraestrutura, podendo adaptá-la para suportar esse tipo de processo;

- No caso de uma migração entre uma nuvem privada e um provedor público, o controle em termos de infraestrutura é menor. Provedores públicos geralmente atendem a um grande número de clientes, e não costumam ter interesse em modificar sua infraestrutura para suportar migrações de aplicações para *datacenters* de outras companhias. Apesar disso, na nuvem privada ainda existe o controle sobre a infraestrutura, e há a possibilidade de se adaptá-la à infraestrutura do provedor público envolvido na migração. Ainda assim, essa adaptação não garantiria a compatibilidade com todos os provedores públicos, visto que cada um utiliza sua própria solução de infraestrutura, tecnologia de virtualização, etc.

Tendo em mente agora o caso em que um cliente de um provedor de IaaS público deseje migrar sua aplicação para outro provedor público diferente, o cliente não possui controle sobre nenhum recurso de infraestrutura além do que lhe é ofertado como serviço. Provedores públicos como Amazon, Rackspace, Terremark, HP, entre outros, oferecem serviços variados de infraestrutura a seus clientes. Esses provedores, entretanto, oferecem cada qual o seu conjunto específico de serviços de provisionamento de infraestrutura. Dessa forma, nem sempre um serviço oferecido por um provedor é oferecido por outros. Um desses serviços, entretanto, é comum à maioria dos provedores, e é considerado o serviço mais básico de IaaS. Este consiste na instanciação sob demanda de máquinas virtuais, e geralmente oferece aos clientes:

- Controle de uma MV em nível de superusuário (Administrador de Sistema, ou *root*), que garante ao cliente a liberdade de configuração e instalação de qualquer *software* a partir do S.O.;

- um endereço IP público acessível na Internet, que garante acesso à MV de qualquer lugar do mundo, tanto para o cliente realizar suas configurações quanto para os usuários finais acessarem as aplicações do cliente.

É importante ressaltar que apesar de serem MVs instanciadas, os provedores públicos não oferecem aos clientes nenhum controle sobre os hipervisores subjacentes. Em muitos casos não é possível sequer saber qual é o hipervisor utilizado. Boa parte dos provedores oferece além desses pontos, recursos para que os usuários configurem regras de *firewall* através de uma interface externa à MV, principalmente para facilitar a aplicação das regras a múltiplas

MVs instanciadas. Apesar disso, a configuração de rede das MVs instanciadas é limitada às configurações de rede do *datacenter* onde elas são instanciadas, e o tráfego é sujeito às suas regras de segurança. Não há qualquer controle de outros pontos da rede, como por exemplo configuração de roteadores.

O projeto da arquitetura proposta neste trabalho toma como base este cenário: a migração da aplicação de um cliente de um provedor de IaaS público para outro provedor público diferente. Conhecidas todas as restrições ao controle dos recursos de infraestrutura impostas neste cenário, acredita-se que, em geral, se a arquitetura proposta é capaz de auxiliar uma migração neste caso, então ela também é aplicável a outros cenários com o mesmo nível de restrições, ou menores.

3.2 Objetivos da Proposta

Baseando-se na motivação deste trabalho e na discussão da seção anterior, são traçados os objetivos principais desta proposta:

- Dar suporte, através do redirecionamento de fluxos de rede, à migração de aplicações entre diferentes *datacenters* de provedores de IaaS;
- Minimizar o tempo de interrupção de serviço durante o processo de migração, o eliminando quando possível;
- Reduzir, quando possível, a duração do processo de migração em si;
- Propor uma solução baseada em *OpenFlow*, porém que não requeira nenhum equipamento específico para seu funcionamento, tampouco necessite de elementos *OpenFlow* em outro ponto da rede, a não ser em suas bordas;
- Ser transparente para os clientes e para as aplicações migradas;
- Ser flexível, ou seja, que possa ser utilizada em diferentes cenários de migração, com diferentes níveis de controle de recursos de infraestrutura.

3.3 Componentes da arquitetura proposta

A arquitetura proposta é projetada para permitir o redirecionamento de fluxos de rede em uma ampla variedade de provedores IaaS, mas a princípio entre provedores públicos. Como nesse cenário ocorre a tarifação pelo uso dos recursos contratados, um objetivo secundário da

arquitetura é ser econômica nesse aspecto. A arquitetura, ilustrada na Figura 3.2, é composta por cinco elementos: o *Switch* Virtual, o Controlador *OpenFlow*, o cliente de DNS dinâmico, o servidor de DNS dinâmico e o gerente de redirecionamento. Todos os componentes são independentes entre si, e podem ser facilmente instalados tanto em máquinas convencionais na Internet, como em MVs dentro de provedores IaaS, e requerem poucos recursos de *hardware* para seu funcionamento.

3.3.1 *Switch* Virtual

O *Switch* Virtual é um componente de *software* instalado na máquina cuja aplicação será migrada e na máquina para onde a aplicação será migrada. Ele implementa o protocolo *OpenFlow* e possui funcionalidade similar a um *switch OpenFlow* físico. Nas máquinas em que é instalado, ele é configurado de modo a intermediar todos os fluxos de rede que entram e saem delas. Logicamente, tudo ocorre como se um *switch OpenFlow* físico estivesse conectado entre a máquina e seu *gateway*, como mostra a Figura 3.1.

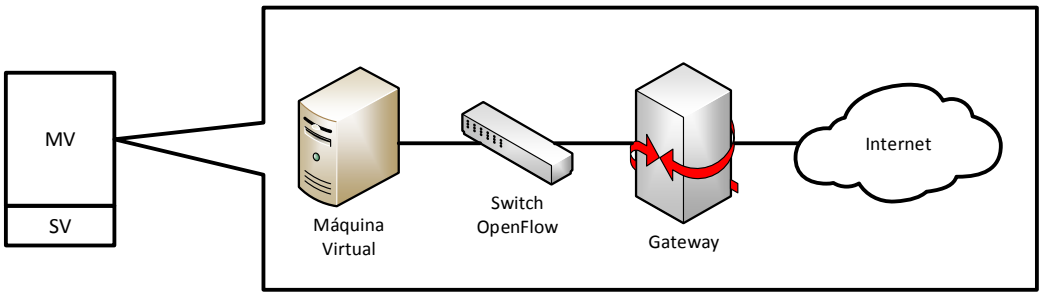


Figura 3.1: Representação lógica do funcionamento do *Switch* Virtual.

3.3.2 Controlador *OpenFlow*

O controlador *OpenFlow* é um processo executando em um servidor. Tal servidor pode ser instalado em qualquer máquina acessível na Internet, bastando ter um endereço IP público. Ele implementa a lógica de controle que coordena o redirecionamento dos fluxos, e os *switches* virtuais das máquinas envolvidas no processo se conectam a ele antes do início do processo. É importante ressaltar que os *switches* virtuais só necessitam se conectar a ele durante o processo de redirecionamento.

3.3.3 DNS Dinâmico (cliente e servidor)

Além dos elementos *OpenFlow*, a arquitetura proposta possui um servidor de DNS dinâmico e seu respectivo cliente. Os clientes da aplicação em trânsito descobrem o IP de seu servidor através de seu nome de domínio, resolvido por DNS. Quando ocorre a migração, o acesso à aplicação passa a ser feito através de um IP pertencente à rede do novo sítio. Para que os novos clientes da aplicação consigam acessá-la é necessário conhecer seu novo IP, e isso é feito através da atualização de entradas em servidores DNS. Devido a sua estrutura global e distribuída, a alteração de uma entrada em um servidor DNS pode demorar horas, ou até dias, para ser propagada por toda a Internet. Até que isso ocorra, o redirecionamento de fluxos é iniciado para evitar a perda de continuidade de serviço. Servidores DNS dinâmicos possuem tempos de cache menores e podem ser atualizados dinamicamente (VIXIE, 1997) através de um cliente, sem a necessidade de alterar arquivos de configuração ou reiniciar serviços. Dessa forma, as mudanças se propagam muito mais rapidamente, reduzindo assim o tempo de redirecionamento de fluxo necessário. Nessa arquitetura, o servidor DNS dinâmico possui os mesmos pré-requisitos que o controlador *OpenFlow*: necessita apenas ser instalado em uma máquina com um IP público acessível na Internet. Nas máquinas de onde a aplicação será migrada e para onde ela será migrada instalam-se um cliente de DNS dinâmico que se conecta ao servidor e atualiza a entrada de nome de domínio com o novo endereço IP. Nesse caso é possível utilizar o servidor de DNS especificamente no momento da migração ou utilizá-lo sempre. Caso ele sempre seja usado, é preciso atualizá-lo com o endereço IP da máquina na origem, quando a aplicação é instalada pela primeira vez.

3.3.4 Gerente do redirecionamento

Por fim, há um elemento que gerencia todo o processo de redirecionamento. Ele, como os outros, pode ser executado a partir de qualquer máquina acessível na Internet. Ele envia comandos aos *switches* virtuais para que estes se conectem ao controlador *OpenFlow* no início do processo, e para que se desconectem ao final do mesmo. Ele também envia comandos aos clientes de DNS dinâmico para que estes atualizem as entradas do servidor DNS dinâmico. Também é responsável por enviar o comando ao controlador *OpenFlow* para que este inicie o processo de redirecionamento de fluxos.

3.4 Visão geral da arquitetura

Em linhas gerais, a arquitetura proposta se organiza como ilustrado pela Figura 3.2. Os *switches* virtuais executando nas máquinas de origem e destino, instruídos pelo gerente, conectam-se ao controlador *OpenFlow* para realizarem o redirecionamento dos fluxos de rede. Durante este processo os fluxos referentes às requisições dos clientes que chegam na máquina na origem são redirecionados para a máquina no destino, que atende as requisições. Sob o ponto de vista dos clientes, entretanto, tudo ocorre de maneira transparente: eles não tomam conhecimento da migração da aplicação ou do redirecionamento de suas requisições.

Neste trabalho foram desenvolvidas duas técnicas diferentes para o redirecionamento dos fluxos, que utilizam os mesmos componentes, porém se comunicam de formas distintas. Tais técnicas são detalhadas nas Seções 3.6.3.1 e 3.6.3.2. As máquinas também possuem os clientes do DNS dinâmico, que se comunicam com seu respectivo servidor para atualizarem entradas do DNS em determinados momentos do processo. Todos os elementos comunicam-se entre si exclusivamente através da Internet.

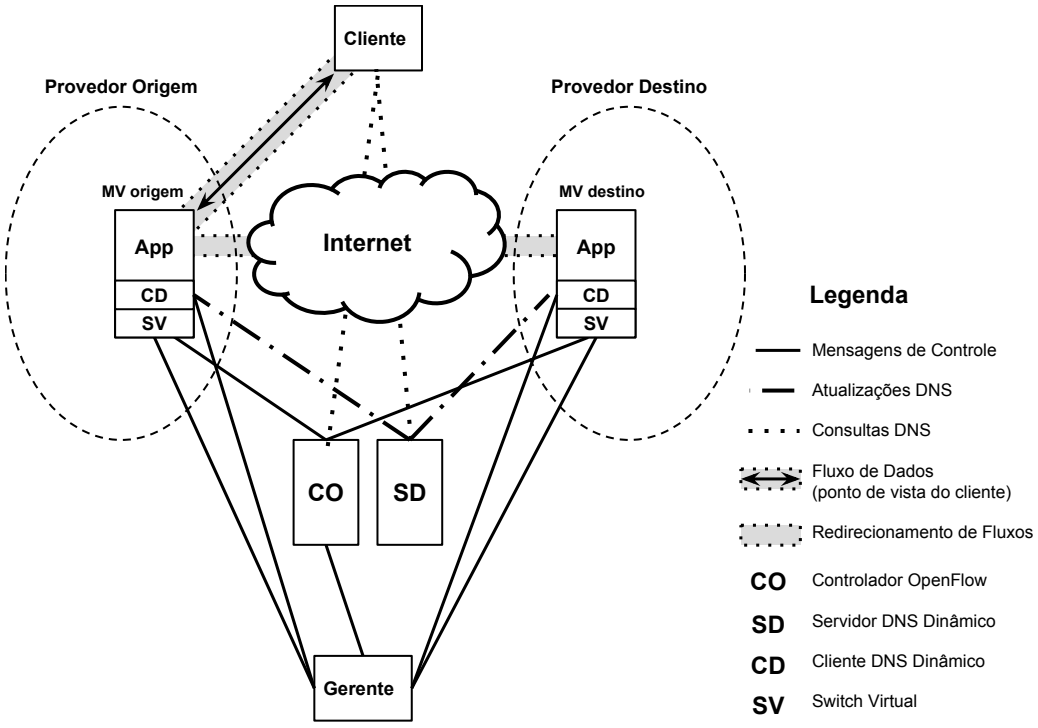


Figura 3.2: Visão geral da arquitetura proposta.

3.5 Integração com outros mecanismos

O processo de migração de uma aplicação envolve duas etapas fundamentais: transferir os dados da aplicação (arquivos executáveis, arquivos de configuração, etc) da origem para o destino e notificar os clientes a respeito de sua nova localização. A arquitetura proposta no presente trabalho tem como objetivo auxiliar esse processo através do redirecionamento de fluxos de rede, permitindo à aplicação continuar atendendo requisições de clientes durante e após a migração. Apesar disso, ela não realiza a migração propriamente dita, pois não se encarrega da transferência de dados. Ela, entretanto, foi projetada para trabalhar em conjunto com um ou mais mecanismos que realizem essa tarefa.

Para atender a clientes durante a migração, a arquitetura assume que nesse intervalo a aplicação atenderá a clientes tanto no sítio de origem como no sítio de destino. Para que isso ocorra é necessário que ela esteja executando nos dois sítios, e que, durante esse período, os dados dessas duas instâncias da aplicação sejam mantidos em um estado consistente, ou seja, os eventuais problemas de sincronização decorrentes do acesso simultâneo a dados em ambos os sítios devem ser tratados.

No cenário tomado como base pelo projeto da arquitetura (migração entre diferentes provedores de IaaS públicos) é possível criar uma instância da aplicação no sítio destino através da instanciação de uma cópia da MV que executa a aplicação no sítio origem, ou através da configuração da aplicação e dos componentes da arquitetura em uma MV instanciada no sítio destino. Para manter o estado consistente dos dados durante o processo de migração podem ser utilizados diferentes mecanismos, como:

- O uso de um sistema de arquivos distribuído entre as máquinas na origem e no destino, que mantenha o acesso aos arquivos sincronizado;
- O uso de um serviço de armazenamento de blocos (*block storage*), como (Amazon.com, Inc., 2014a), configurado de forma que as máquinas na origem e no destino compartilhem o acesso aos mesmos dados, como ilustrado na Figura 3.3 mantendo assim seu acesso sincronizado;
- O uso de ferramentas de cópia sincronizada, como *rsync* (TRIDGELL, 1999);
- O uso de mecanismos de transferência de arquivos baseados em deduplicação, como o apresentado no trabalho de Hirofuchi et al. (HIROFUCHI, 2009), que pode ser adaptado para a migração entre provedores IaaS públicos, através da instanciação de uma segunda

MV em cada provedor para abrigar os dados referentes à aplicação, bem como o mecanismo de transferência, como ilustrado na Figura 3.4. Assim, as MVs com o mecanismo realizariam a transferência e manteriam o acesso sincronizado.

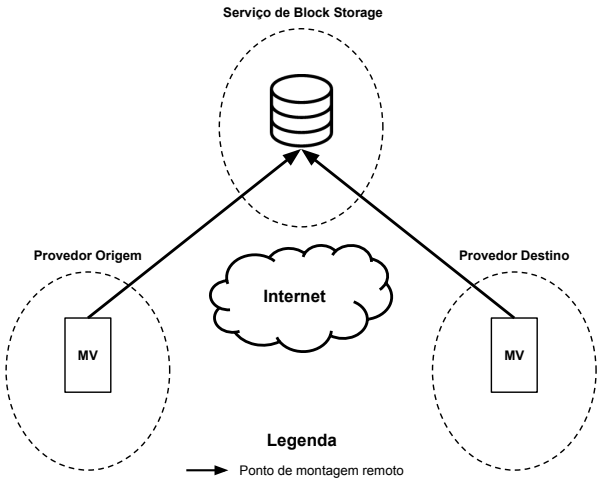


Figura 3.3: Exemplo de mecanismo para manter a consistência dos dados da aplicação através de serviço de *Block Storage*

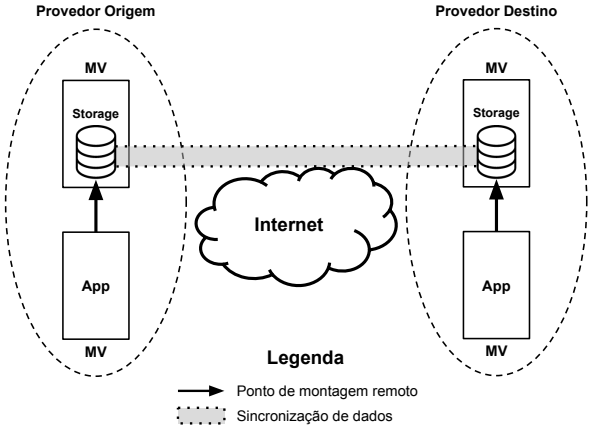


Figura 3.4: Adaptação do mecanismo de Hirofuchi et al. Os dados da aplicação são armazenados em outra MV e montados pelas MVs que executam a aplicação. Essa estrutura de duas MVs é replicada no sítio destino, e as MVs que armazenam os dados se comunicam entre si através do mecanismo e operações de sincronização de dados de disco para manterem a consistência dos dados da aplicação.

É importante ressaltar que esses mecanismos tratam apenas da manutenção da consistência de dados de disco. A arquitetura, entretanto, foi projetada flexível o suficiente para ser utilizada em conjunto com qualquer mecanismo que garanta essa consistência para dados de memória.

3.6 Exemplo do funcionamento da arquitetura em uma migração de aplicação

Para o entendimento completo da arquitetura proposta, as seções seguintes são dedicadas a mostrar um exemplo de como se dá o funcionamento da mesma durante uma migração. É importante enfatizar que os mecanismos citados na Seção 3.5 foram exibidos como exemplo. Qualquer outro mecanismo que realize a tarefa de manter um estado consistente dos dados da aplicação durante a migração poderia ser utilizado. Os detalhes específicos do funcionamento de cada um, bem como detalhes de sua integração com a arquitetura aqui proposta e a migração da aplicação em si estão fora do escopo desta pesquisa. No processo de migração descrito a seguir assume-se que:

- Uma cópia da MV com a aplicação em migração é instanciada no sítio destino (bem como todos os componentes da arquitetura já instalados e configurados);
- Os componentes da arquitetura já estão instalados e configurados previamente, na MV origem, em sua cópia no destino e em outras máquinas espalhadas pela Internet;
- Todas as máquinas estão configuradas com endereços IP públicos e são acessíveis a partir da Internet;
- Um dos mecanismos citados, ou outro equivalente, está sendo utilizado para realizar a transferência dos dados da aplicação e manter consistente o seu estado durante esse processo.

3.6.1 Estado inicial

Seja a migração de uma aplicação no cenário descrito na Seção 3.1. Inicialmente tem-se a aplicação executando em uma MV instanciada no provedor origem. Nesse ponto, a aplicação atende requisições da maneira usual. A Figura 3.5 ilustra a interação entre os agentes em uma requisição típica. O cliente consulta o servidor DNS, resolve o endereço IP da máquina na origem e começa a fazer as requisições, que são diretamente atendidas. Nesse momento, apesar de estar interceptando todos os fluxos que chegam e saem da máquina, o *switch* virtual apenas os encaminha, funcionando como uma interface de rede convencional.

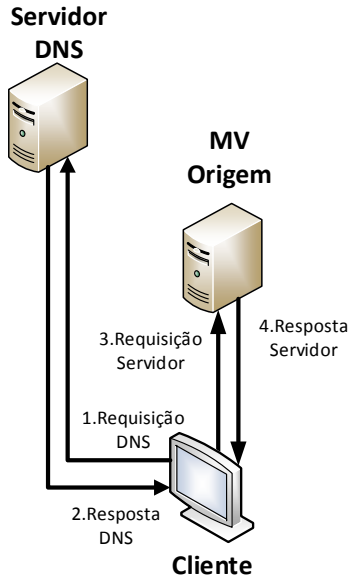


Figura 3.5: Estado inicial: a origem atende as requisições diretamente.

3.6.2 Início da migração

Para iniciar o processo de migração da aplicação são necessárias as seguintes etapas:

1. instanciar a cópia da MV no destino;
2. iniciar a aplicação na cópia;
3. iniciar o processo de sincronização de dados;
4. iniciar o redirecionamento de fluxos de rede.

Como neste trabalho é exibida apenas a proposta da arquitetura para o redirecionamento de fluxos, nessa seção serão descritos apenas os detalhes dela em relação a essa etapa. Assume se aqui que os primeiros três passos são executados corretamente e em ordem. Para iniciar o processo de redirecionamento de fluxos é necessário acionar o gerente de migração, para que coordene o processo enviando mensagens aos outros elementos. Ele envia comandos para os *switches* virtuais nas MVs origem e destino, ao cliente do DNS dinâmico no destino e ao controlador *OpenFlow*. Os *switches* recebem um comando para se conectarem ao controlador *OpenFlow*. O cliente do DNS recebe um comando para atualizar o servidor DNS dinâmico. Por fim, o controlador *OpenFlow* recebe um comando para que o redirecionamento de fluxos seja efetivamente iniciado. A Figura 3.6 exibe esses comandos e ordem em que eles ocorrem.

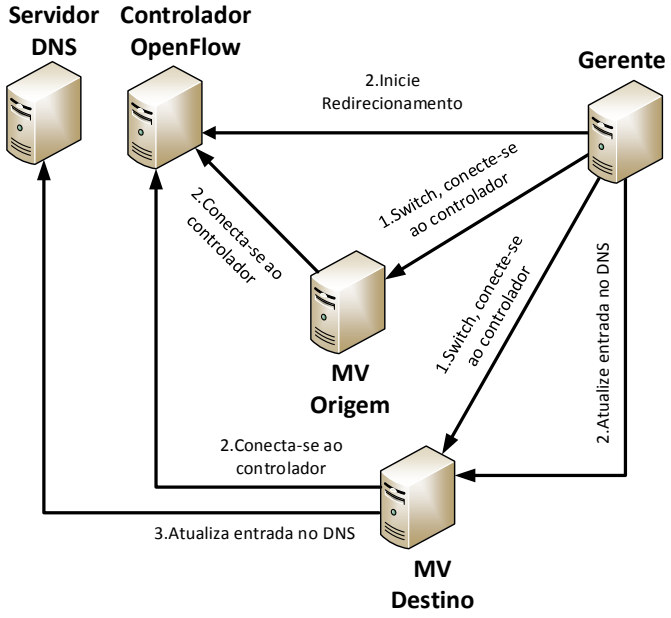


Figura 3.6: Comandos enviados pelo gerente de migração para iniciar o redirecionamento de fluxos.

3.6.3 Redirecionamento de fluxos

Durante a migração da aplicação, o redirecionamento de fluxos deve ocorrer para que a aplicação continue atendendo a requisições de clientes enquanto seus dados são transferidos da origem para o destino e enquanto os clientes não são todos notificados do novo endereço da aplicação. Dessa forma, ele deve ocorrer durante todo o tempo em que o mecanismo de migração estiver realizando a sincronização dos dados da aplicação entre as MVs origem e destino, e durante o tempo necessário para a total propagação da alteração feita no servidor de DNS dinâmico. Os fluxos iniciados antes do início do redirecionamento continuam a ser atendidos pela aplicação na MV origem. Os fluxos iniciados a partir do início dessa etapa são redirecionados para a MV no destino, cuja aplicação passa a atender as requisições. Como discutido na seção 3.5, esta estratégia de redirecionamento pressupõe que o mecanismo de transferência dos dados da aplicação utilizado em conjunto com a arquitetura proposta mantem consistente o acesso aos dados da aplicação nesse momento.

Uma vantagem desta estratégia de redirecionamento é ela poder ser utilizada para reduzir o tempo necessário para a transferência dos dados da aplicação. A MV na origem atende apenas os fluxos iniciados até o instante em que se inicia o redirecionamento. Quando estes se encerram, ela não atende mais nenhuma requisição, nenhum dado da aplicação é modificado nela a partir desse momento. Como, ao final do processo de migração a MV na origem tende a ser

encerrada, a consistência dos dados da aplicação entre origem e destino só precisa ser mantida enquanto a aplicação na origem ainda atende a requisições. O mecanismo utilizado para transferir os dados e manter o estado consistente pode ser configurado ou adaptado para se aproveitar disso.

Neste trabalho foram desenvolvidas duas versões para esta etapa de redirecionamento de fluxos. A descrição das mesmas e a justificativa para o desenvolvimento de ambas são exibidas a seguir.

3.6.3.1 Primeira versão

A primeira versão procura realizar o redirecionamento através do menor caminho. Ela é ilustrada na Figura 3.7 através de um exemplo de atendimento de uma requisição. O cliente consulta o servidor DNS dinâmico, recebe o endereço IP da máquina servidora da aplicação na origem e faz uma requisição a ela. Ao chegar no *switch* virtual dessa máquina ele, orientado pelo controlador *OpenFlow*, redireciona o fluxo para a máquina no destino através da substituição do campo "IP destino" no cabeçalho dos pacotes. Ao chegar no *switch* virtual da máquina servidora no destino, este o encaminha para a aplicação, que atende a requisição e gera uma resposta. O fluxo correspondente a essa resposta passa novamente pelo *switch* virtual e também é alterado. O valor do campo "IP origem" do cabeçalho dos pacotes desse fluxo é alterado, e passa a receber o valor do IP da máquina servidora na origem. Quando o cliente recebe a resposta de sua requisição, ela vem da máquina servidora no destino, porém para este é como se ela tivesse sido enviada pela máquina servidora na origem. É importante destacar que, como a rede é baseada em *OpenFlow*, a comunicação dos *switches* virtuais com o controlador *OpenFlow* (passos 4, 5, 7 e 8 da figura) ocorrem apenas com o primeiro pacote de cada fluxo novo. Os pacotes subsequentes são redirecionados diretamente (passos 3, 6 e 9).

3.6.3.1.1 O problema dos fluxos novos

Durante a fase de redirecionamento, entretanto, servidor DNS dinâmico já foi atualizado e o endereço atual do servidor está se propagando nas novas consultas. Logo, os clientes passam a conhecer o novo IP da aplicação e a fazer requisições diretamente ao novo endereço. Dessa forma, o *switch* virtual no destino precisa identificar esses fluxos e diferenciá-los dos fluxos dos clientes que foram redirecionados, os encaminhando sem modificações. Para realizar essa identificação foi utilizada a seguinte estratégia, ilustrada na Figura 3.9.

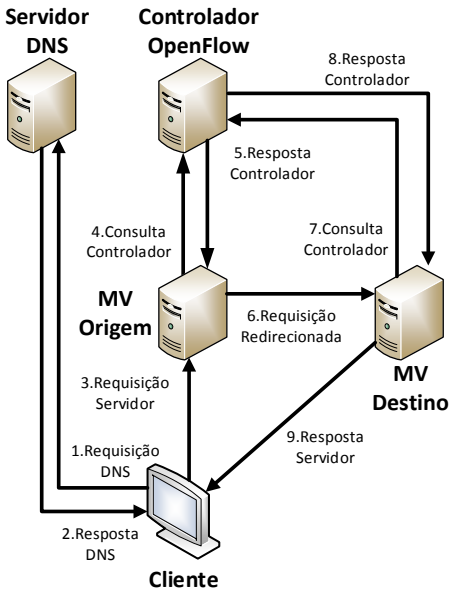


Figura 3.7: Comunicação entre os elementos da arquitetura na 1ª versão do redirecionamento de fluxos.

O controlador *OpenFlow* possui uma tabela com entradas como a mostrada na Figura 3.8, que mapeia pares (*endereço IP, porta origem*) de cada fluxo em um valor numérico, único, entre 0 e 65535, que o identifica [1]. Quando o primeiro pacote do fluxo chega no *switch* virtual da origem, ele o envia ao controlador, que armazena o par (IP, porta) em sua tabela, atribuindo a ele um número de identificação **N**. Ele então instrui o *switch* virtual a tratar do pacote da seguinte forma: substituir os valores dos campos **IP destino**, com o IP da máquina no destino, **IP origem**, com o valor do IP da máquina na origem e **porta origem**, com o valor **N**, e encaminhá-lo para o novo destino.

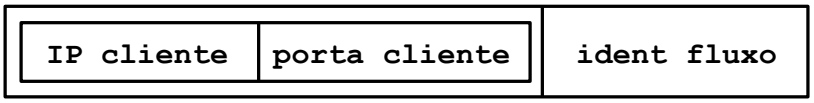


Figura 3.8: Estrutura de dados que identifica os fluxos dos clientes.

Essa modificação torna os pacotes redirecionados claramente diferentes dos fluxos novos. Todos eles têm como origem o endereço da máquina na origem. Ao chegarem no *switch* virtual do destino, os fluxos são encaminhados à aplicação e respondidos normalmente. Os fluxos de resposta, entretanto, também passam pelo *switch* virtual e dessa vez são alterados, da seguinte

[1] Os valores precisam estar nesse intervalo pois são inseridos no campo porta origem dos pacotes, que possui tamanho de 16 bits.

forma: O primeiro pacote de cada fluxo é enviado ao controlador, que inspeciona os campos **IP destino** e **porta origem**. Todos os pacotes que chegarem no *switch* virtual do destino endereçados ao IP da máquina na origem, na porta da aplicação migrada são respostas de fluxos redirecionados. O controlador procura pelo número identificador do fluxo (que se encontra no campo **porta destino**) em sua tabela, encontra os dados do respectivo cliente, e instrui o *switch* virtual a alterar esse e os próximos pacotes do fluxo da seguinte forma: substituir os campos **IP origem** com o IP da máquina na origem, **IP destino** com o IP do cliente encontrado na tabela, e **porta destino** com o valor da porta encontrado na tabela. Assim, os pacotes saem do destino como se tivessem sido enviados pela origem, tornando o redirecionamento transparente para os clientes, sem problemas como quebra de *sockets*. Além disso, os fluxos dos clientes que já conhecem o novo endereço da aplicação agora são facilmente identificados: fluxos cujo campo **IP origem** com valores diferentes do IP da máquina na origem e com o valor da porta da aplicação no campo **porta destino** são clientes novos, são encaminhados para a aplicação; os respectivos fluxos de respostas são identificados e tratados da mesma maneira.

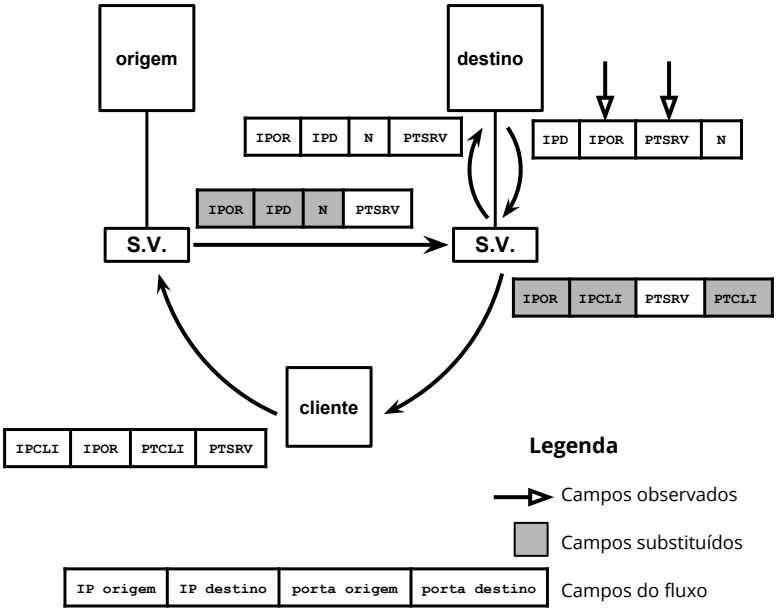


Figura 3.9: Substituição de campos e encaminhamento de pacotes na primeira versão do redirecionamento de fluxos. As setas indicam o encaminhamento do fluxo, e as estruturas mostram os campos de cabeçalhos principais referentes aos fluxos.

3.6.3.2 Segunda versão

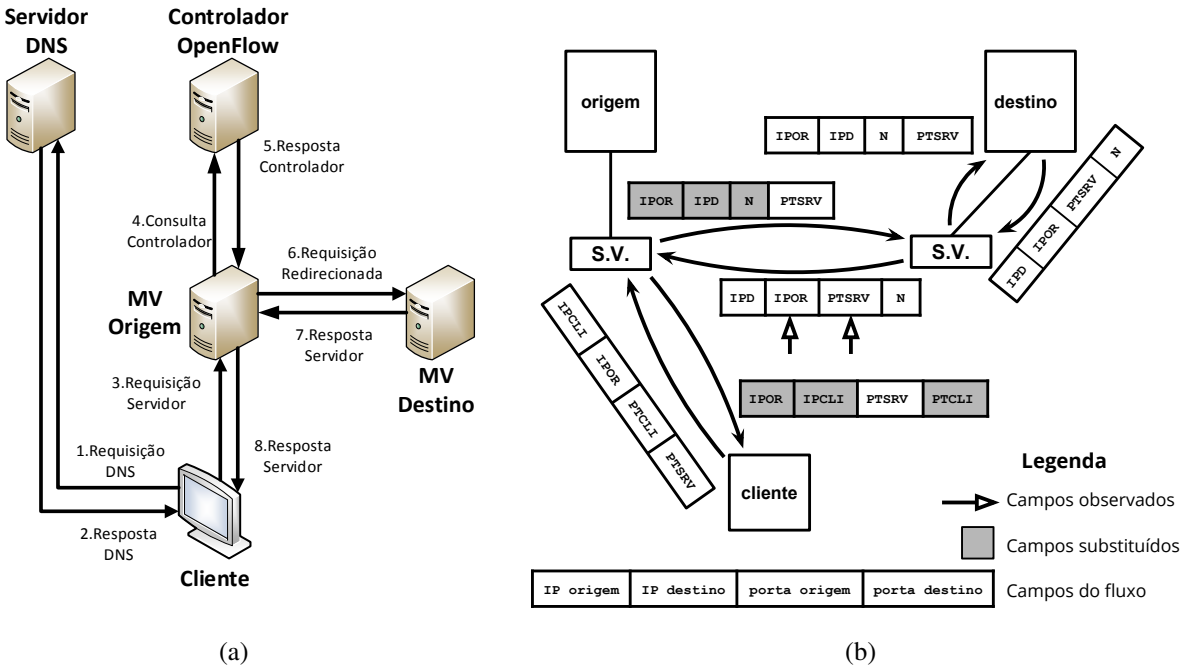
A primeira versão do redirecionamento de fluxos foi desenvolvida e posteriormente testada com sucesso em um ambiente de rede virtual simulando uma rede de longa distância executando algoritmos padrão de roteamento da Internet. Sua implantação, entretanto, não é factível

na infraestrutura atual da Internet porque poderia ser confundida com uma técnica maliciosa conhecida como *IP Spoofing* (EHRENKRANZ; LI, 2009). Tal técnica, utilizada há anos em atividades maliciosas na Internet, consiste em alterar o valor do campo IP origem de pacotes para ocultar a verdadeira localização de um atacante. *IP Spoofing* ainda hoje é utilizada nos mais variados tipos de ataques, como por exemplo DDOS. Ao longo dos anos foram criados diversos mecanismos de segurança buscando impedir essa prática. Como consequência, a grande maioria dos roteadores e *firewalls* utilizados hoje em dia na Internet implementam por padrão mecanismos para se defender dessa ameaça.

A implicação direta disso é que, apesar dos protocolos de roteamento em si não checarem o campo IP origem dos pacotes, os roteadores e *firewalls* o fazem, e descartam quaisquer pacotes que se assemelhem a esses pacotes maliciosos. Como visto na Seção 3.6.3.1.1, na primeira estratégia de redirecionamento, o campo **IP origem** dos pacotes é alterado no *switch* virtual do destino antes de serem encaminhados para o cliente. Quando esses pacotes alterados chegam na borda da rede onde estão, como um roteador, ele constata que o endereço de origem dos pacotes não pertence à rede de onde ele está vindo, e automaticamente o descarta. Sendo assim, para que o redirecionamento de fluxos pudesse funcionar corretamente na infraestrutura atual da Internet, mantendo a premissa de nenhum controle a não ser o interno às MVs e sem nenhum outro elemento *OpenFlow* na rede a não ser nos servidores da aplicação em migração, foi necessário desenvolver uma segunda versão do redirecionamento de pacotes.

Para que os pacotes sejam encaminhados corretamente ao saírem do *switch* virtual, eles necessariamente precisam conter no campo **IP origem** um endereço IP pertencente à rede de onde estavam sendo enviados ou seriam descartados. Entretanto, a estratégia utilizada na primeira versão para diferenciar os fluxos redirecionados dos fluxos novos faz exatamente isso: todos os fluxos redirecionados são modificados e saem da origem com o endereço IP da origem nesse campo. Por essa razão a primeira parte do redirecionamento permanece inalterada na segunda versão, como pode-se observar na Figura 3.10(a) (os passos 1 a 6 são idênticos à primeira versão) e na Figura 3.10(b) (os campos modificados no caminho de ida são os mesmos). Ao chegarem no *switch* virtual do destino, tais fluxos precisam ser respondidos, mas os fluxos de resposta só podem ter valores válidos no campo **IP origem** dos pacotes. Com as restrições impostas pelo cenário de aplicação, não há outro ponto que possa ser controlado pela rede *OpenFlow* a não ser os *switches* virtuais. Por esse motivo, optou-se por simplesmente encaminhar os fluxos para serem respondidos e as respostas encaminhadas de volta à origem.

Os fluxos, então, são redirecionados ao destino, respondidos, e voltam para a origem (passo 7 da Figura 3.10(a)). Nesse ponto é preciso identificar os remetentes originais de cada requisição



(a) (b)

Figura 3.10: Comunicação entre agentes e reescrita de pacotes na segunda versão do redirecionamento de fluxos.

para encaminhar o fluxo de resposta correspondente a cada um deles. Entretanto, a tabela implementada na primeira versão fazia exatamente isso: cada fluxo é identificado univocamente através do identificador (**N**, na Figura 3.10(b)). Dessa forma, a segunda etapa de redirecionamento passa a ser realizada também no *switch* virtual da origem (passo 8 da Figura 3.10(a)).

3.6.4 Final da migração

O processo de migração se encerra quando os dados de disco entre origem e destino estão sincronizados e todas as requisições feitas ao DNS retornam o endereço IP da máquina no destino. Nesse ponto, as requisições são atendidas como exibido na Figura 3.11, ou seja, da mesma forma que eram atendidas antes da migração. Nesse ponto, o *switch* virtual da máquina destino pode ser desconectado do controlador *OpenFlow*, e passa a atuar como uma interface de rede convencional. A máquina virtual na origem pode então ser desligada.

3.6.5 Comentários sobre as duas versões do redirecionamento

Sobre as técnicas de redirecionamento de fluxos desenvolvidas, podem ser feitos alguns comentários. Em primeiro lugar, pode-se notar que na segunda versão,o *switch* virtual da máquina no destino apenas encaminha fluxos, trabalhando como uma interface de rede comum. Como

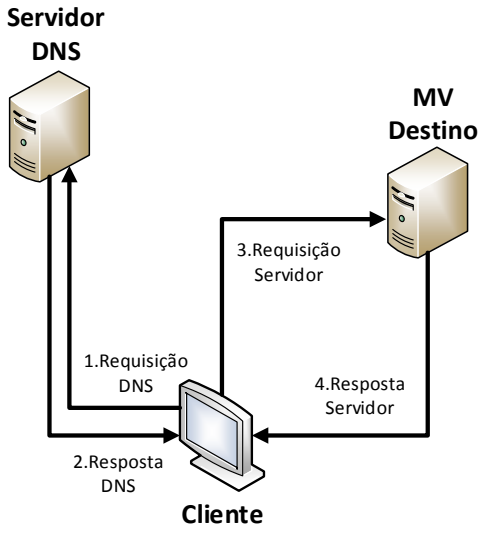


Figura 3.11: Estado final: o destino passa a atender as requisições diretamente.

esse é o comportamento padrão do *switch* virtual, a segunda versão não requer que ele esteja conectada ao controlador *OpenFlow*. Todas as operações de reescrita de campos de cabeçalho necessárias ao redirecionamento são feitas pelo *switch* virtual da origem. Sendo assim, a segunda versão torna opcional a instalação de um *switch* virtual no destino, tornando a abordagem mais flexível nesse aspecto. Mesmo assim, no cenário descrito, a presença desse componente na máquina destino ainda é interessante, visto que com ele é possível posteriormente migrar a aplicação do destino para outra localidade.

Outro detalhe da segunda versão, e talvez o mais importante, é que os pacotes redirecionados agora percorrem um caminho mais longo do que na primeira versão. Além disso, os pacotes fazem duas vezes o trajeto entre origem e destino. Isso faz com que o impacto do redirecionamento seja maior dependendo da distância entre origem e destino, o que pode se tornar, em casos extremos, um fator limitante para a aplicação da abordagem. Apesar disso, a segunda versão do redirecionamento consegue cumprir todos os objetivos propostos: ela garante a transparência de localidade aos clientes, procura reduzir o tempo de sincronização dos dados (como explicado na Seção 3.6.3). e realiza o redirecionamento de fluxos utilizando conceitos de RDS na infraestrutura atual da Internet, sem o uso de nenhum recurso adicional, e em um cenário com restrito controle sobre a infraestrutura. O teste efetuado em ambiente real, que será exibido no Capítulo 4, apresenta alguns resultados sobre o desempenho da abordagem e abre espaço para maiores discussões a esse respeito.

Um terceiro aspecto a ser ressaltado sobre a segunda versão é que nela o *switch* virtual da

origem acaba atuando de maneira semelhante a um *proxy* entre o destino e os clientes, visto que ele intermedeia todas as conexões feitas à aplicação alvo, e por ser baseado em *OpenFlow*, ele se beneficia com suas vantagens. Isso abre espaço para modos diferentes de utilizar essa abordagem, em contextos diferentes, o que ressalta sua flexibilidade.

3.7 Implementação do protótipo funcional

Para implementar a abordagem proposta e testar sua viabilidade foi desenvolvido um protótipo funcional. A seguir são descritos alguns aspectos importantes envolvendo a sua implementação.

3.7.1 Controlador *OpenFlow*

O controlador *OpenFlow* foi implementado na linguagem de programação *Python*, utilizando o POX (NICIRA, INC., 2012), que, assim como o NOX (GUDE, 2008), implementa o conceito de NOS. O POX implementa as mensagens do protocolo *OpenFlow* e fornece uma interface de programação (*Application Programming Interface* - API) baseada em eventos, que permite o desenvolvimento rápido de controladores, e tem sido bastante usado, especialmente para fins de pesquisa. O controlador implementado neste trabalho reage a três eventos da rede *OpenFlow*: *PacketIn*, *FlowMod* e *FlowRemoved*. Além desses, ele reage a um evento personalizado, denominado *Start*.

3.7.1.1 Eventos

O evento de *PacketIn* é disparado quando um pacote chega em algum *switch* da rede e não casa com nenhuma das regras da tabela de fluxos do *switch*. O *switch* então encapsula esse pacote em uma mensagem *OpenFlow* e a envia ao controlador. Dessa forma o controlador identifica todos os fluxos trafegando pela rede conforme eles vão chegando, e os trata de acordo com a lógica descrita na Seção 3.6.3. No controlador implementado, os fluxos são identificados através dos eventos *PacketIn* mas são tratados através de eventos *FlowMod*. Estes são disparados quando o controlador envia uma nova regra para um *switch* incluir em sua tabela de fluxos. Além destes, o controlador implementado também reage ao evento *FlowRemoved*, que é disparado quando uma entrada na tabela de algum *switch* da rede é removida (por ter sido encerrada ou por ter expirado) e faz com que o *switch* em questão envie uma mensagem ao controlador encapsulando a regra que foi removida. O controlador implementado reage a esse evento para atualizar suas estruturas de dados, que são explicadas em detalhe a seguir.

Além desses eventos, que são regulares do *OpenFlow*, o controlador implementado precisa saber em que momento o redirecionamento de fluxos deve ser iniciado. Para isso foi desenvolvido um evento personalizado, chamado de *Start*. Quando esse evento é disparado, o controlador altera o modo como trata os novos fluxos. Seu comportamento inicial é instruir os *switches* a simplesmente encaminharem os fluxos que chegam, como fazem as interfaces de rede comuns. Disparado o evento *Start*, eles passam a realizar o redirecionamento dos fluxos que chegam. A listagem de código 3.1 ilustra a alteração do comportamento a partir do disparo do evento *Start*. As funções *handle_packets_default* e *handle_packets_mig*, que definem os comportamentos antes e durante o redirecionamento são listadas no Apêndice A.

Listagem 3.1: Função que trata dos eventos *Start*.

```
1  def _handle_MigrationStart (self, event):
2    log.info("Evento Start disparado")
3    #removendo funcao que encaminha pacotes
4    core.openflow.removeListener(self.handle_packets_default)
5    #adicionando funcao que redireciona pacotes
6    core.openflow.addListenerByName("PacketIn", self.handle_packets_mig)
```

No protótipo implementado, esse evento é disparado pelo administrador do sistema, mas ele pode ser disparado automaticamente, no caso de um mecanismo integrado de migração. Pensando nisso, procurou-se uma abordagem flexível: o controlador implementado abre um *socket* em uma porta específica. Para iniciar a migração, basta conectar-se a esse *socket* e enviar uma mensagem específica. No protótipo implementado a mensagem é uma *string* simples (no caso, *start*). A listagem de código **??** exibe como o evento *Start* é disparado no protótipo implementado.

Listagem 3.2: Função que cria o *socket* usado para disparar o evento *Start*. label

```
1  def open_socket(self):
2    try:
3      sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4    except socket.error, msg:
5      log.debug("Erro no socket Cod: %s, Msg: %s", str(msg[0]),msg[1])
6      return
7    try:
8      sock.bind(("localhost",PORTA))
9    except socket.error, msg:
10     log.debug("Erro no bind. Cod: %s, Msg: %s", str(msg[0]),msg[1])
11     return
12
13   sock.listen(2)
```

```
14
15  while True:
16    conn, addr = sock.accept()
17    data = conn.recv(1024)
18    log.info("%s", data)
19    if data == MENSAGEM:
20      # evento inicio de migracao
21      self.miglisten.raiseEvent(MigrationStart)
22    else:
23      pass
24    conn.close()
25  sock.close()
```

3.7.1.2 Estruturas de dados internas

Ambas as versões do redirecionamento de fluxos utilizam uma tabela para mapear os fluxos redirecionados. Como cada novo fluxo de cada cliente gera uma entrada nessa tabela, ela pode crescer rapidamente. Apesar da tabela ter um tamanho máximo limitado (65535 entradas), um dos princípios do projeto é implementar uma solução econômica em termos de recursos computacionais. Por isso a tabela foi implementada de modo a otimizar a busca das entradas. Ela é constituída de duas tabelas *hash* (em *Python*, chamadas *dictionaries*) que são atualizadas simultaneamente: uma é indexada pelo número identificador de fluxo e a outra pelo par (IP,porta). Dessa forma, sempre que necessário pesquisar algum dado na tabela, seja ele um identificador ou um par, o tempo de busca se mantém constante independente de seu tamanho.

Além disso, como o número identificador de fluxo é armazenado nos campos de cabeçalho destinados a armazenamento de número de porta do protocolo de transporte, que possui 16 bits, o número tem de estar dentro do intervalo de números possíveis de serem representados nesse espaço, ou seja, de 0 a 65535. Apesar de ser uma quantidade razoável, é preciso gerenciar bem os números nesse intervalo, para evitar ao máximo o esgotamento de todos os números possíveis. Para isso, o controlador mantém uma lista com todos os números que estão sendo usados. Cada vez que um novo fluxo é redirecionado, a lista marca aquele número como utilizado, e a cada vez que um fluxo é encerrado, o que é sinalizado pelo evento de *FlowRemoved*, o número correspondente é marcado na lista como disponível novamente.

3.7.1.3 Rede interna dos provedores

O controlador foi implementado de modo que os *switches* virtuais controlados por ele se adaptassem à rede na qual estivessem inseridos. Para tanto, foi assumida como padrão a configuração de rede mais simples usada pela EC2 da AWS: As instâncias *on-demand* são criadas com uma única interface de rede, configurada com um endereço IP privado, atribuído por DHCP[2]. Com base nisso, o controlador descobre através do protocolo ARP o endereço MAC e o IP privado da MV onde o *switch* está executando, e as mesmas informações de seu *gateway*, para que assim possa montar todos os pacotes necessários. Porém, para que isso ocorra, é preciso que haja comunicação de rede do *gateway* com a MV após o *switch* se conectar ao controlador. Qualquer comunicação externa com a MV, como um *ping* em seu IP público, gera esse tipo de comunicação interna. Há também a possibilidade de, caso já conhecidos esses valores, eles serem passados como parâmetros no momento da inicialização do controlador.

3.7.2 *Switch* virtual e DNS dinâmico

A implementação do *switch* virtual foi feita através do *openvswitch* (OVS) (PFAFF, 2009), uma implementação em *software* de um *switch OpenFlow*. Ele tem sido muito utilizado em conjunto com hipervisores para expandir o controle e a funcionalidade dos mesmos em relação à configuração de rede de MVs. Neste trabalho, que foi implementado em computadores executando Linux, ele foi instalado na máquina alvo como um módulo de *kernel*, e foi configurado como um interface de rede em modo *bridge*, que foi configurada para ser a rota padrão de saída da MV. Dessa forma, todos os pacotes que entram ou saem da MV passam por essa interface. Internamente, o OVS foi configurado como um *switch* de duas portas, para poder diferenciar os fluxos que entram ou saem da MV. O Apêndice A exibe detalhes da instalação e configuração do mó*openvswitch* nas MVs utilizadas.

O servidor DNS dinâmico foi implementado através de um servidor DNS comum, configurado para aceitar atualizações dinâmicas. O cliente foi implementado através de uma ferramenta que implementa as mesmas especificações do servidor e envia comandos de atualização a ele. Esse componente, entretanto, pode ser implementado através de qualquer serviço de DNS dinâmico disponível atualmente, como (No-IP.com, 2014).

[2]Tais informações foram coletadas pela experiência de uso dos serviços da AWS e também através do estudo da plataforma de gerenciamento de nuvens Eucalyptus (Eucalyptus Systems, Inc., 2014), que possui uma implementação muito similar à do EC2

Capítulo 4

TESTE DA ARQUITETURA

Este capítulo descreve os testes realizados ao longo do desenvolvimento da arquitetura proposta para validar seu funcionamento, os testes realizados para a avaliação de seu comportamento em um caso de migração de aplicação, bem como exibe os resultados obtidos e uma análise dos mesmos.

4.1 Teste da 1ª versão do redirecionamento de fluxos

O primeiro protótipo funcional foi desenvolvido para comprovar a viabilidade da primeira estratégia de redirecionamento de fluxos, baseada no menor caminho. Para o teste configurou-se um ambiente de rede virtual utilizando a ferramenta Mininet para simular uma rede baseada em *OpenFlow*. Em conjunto com ela foi utilizado o *RouteFlow* (NASCIMENTO,), para executar protocolos de roteamento reais na rede virtual e simular um ambiente de WAN similar à Internet. A rede simulada possuía roteadores executando o protocolo OSPF e nas bordas máquinas executando um servidor *web* cada, e com os elementos da arquitetura instalados, fazendo o papel das MVs origem e destino. Em outra borda da rede havia uma máquina fazendo o papel do cliente, que realizava requisições contínuas de páginas *web*, utilizando a ferramenta *wget*. Em outras máquinas do ambiente simulado executavam os outros componentes da arquitetura proposta. O processo de redirecionamento foi testado e, através da análise dos arquivos de *log* dos servidores *web* comprovou-se o funcionamento da primeira versão do redirecionamento. Isso provou que o redirecionamento pelo menor caminho é factível em uma Internet com menores restrições de segurança.

4.2 Testes da 2ª versão do redirecionamento de fluxos

O segundo protótipo funcional desenvolvido implementa a segunda estratégia de redirecionamento de fluxos, que leva em consideração o problema de IP *Spoofing*. A princípio foram realizados testes para validar o seu funcionamento, utilizando uma infraestrutura de nuvem privada local, um cenário não tão controlado quanto um ambiente simulado, mas com maior controle do que um ambiente de nuvem pública, e já com a presença de *firewalls* para testar o funcionamento da abordagem com restrições de segurança. Comprovado o seu funcionamento, foram realizados testes utilizando um ambiente de nuvem pública, para avaliar o comportamento na Internet.

4.2.1 Teste com nuvens privadas

Para o teste com nuvens privadas foram utilizados os recursos de infraestrutura do DC-UFSCar: a rede de produção do departamento e as máquinas de um *cluster*. Cada máquina possui dois processadores AMD Opteron de 2 GHz cada, 8GB de memória RAM e 1TB de disco rígido. Dois conjuntos de máquinas foram configurados como duas nuvens IaaS privadas separadas através do controlador de nuvem *Eucalyptus* (Eucalyptus Systems, Inc., 2014). Cada nuvem foi configurada em uma sub-rede diferente, estando as nuvens separadas por dois saltos (*hops*), cada qual com seu *firewall* e mecanismos de segurança padrão habilitados. A topologia desse cenário é ilustrada na Figura 4.1. Neste cenário foi instanciada em cada nuvem uma MV, com os elementos da arquitetura instalados e com um servidor *web*. A máquina cliente executava uma versão modificada da ferramenta *Apache Benchmark* (ab), que fazia requisições GET não persistentes a uma página *index.html* contínuas e concorrentes.

4.2.1.1 Resultados

o gráfico exibido na Figura 4.2 mostra o resultado de um dos testes realizados. O teste teve duração total de 45 segundos, e o ab procurava manter um nível de concorrência de 5 requisições simultâneas, ou seja, fazia requisições em uma velocidade que procurava manter sempre 5 conexões abertas com o servidor simultaneamente. Nos primeiros 15 segundos de teste não há redirecionamento, e as requisições são atendidas diretamente pela origem. A partir de aproximadamente 15 segundos após o início do teste, é iniciado o redirecionamento de fluxos, e as requisições passam a ser redirecionadas conforme a segunda estratégia de redirecionamento de fluxos, e passam a ser atendidas pelo destino. O redirecionamento é mantido por mais 10 segundos, e por volta de 25 segundos após o início o teste, o servidor DNS dinâmico é atualizado.

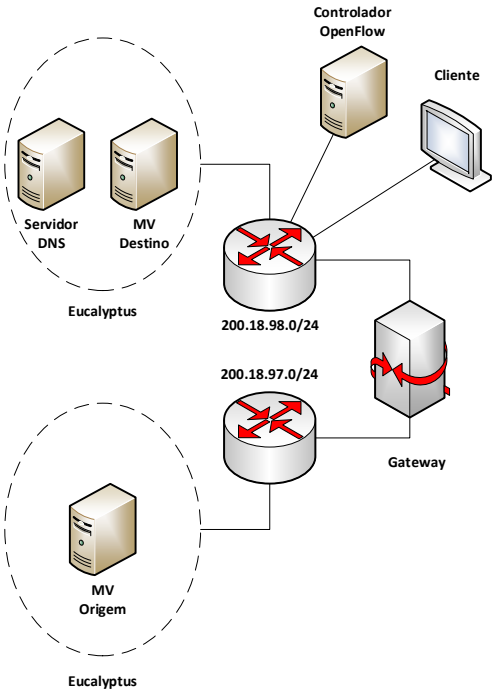


Figura 4.1: Topologia do cenário de testes com nuvens privadas.

A partir desse ponto, as requisições passam a ser atendidas diretamente pelo destino, e isso continua até o fim do teste. Sobre o gráfico, é interessante notar que como o caminho percorrido é maior, o tempo de resposta médio cresce, e o número de requisições atendidas por tempo diminui, o que o deixa mais escasso. Como a máquina cliente se encontrava na mesma rede local que a MV na nuvem destino, o tempo de resposta diminui, em média, após a migração.

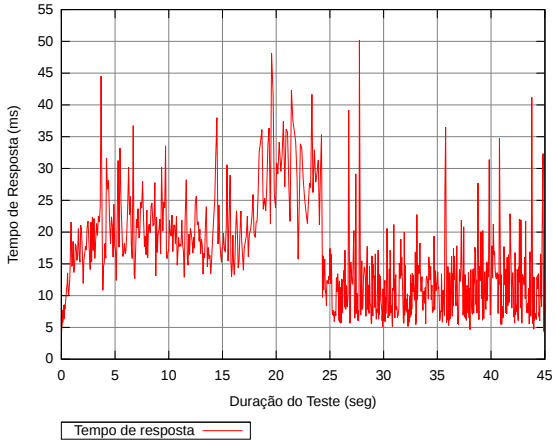


Figura 4.2: Gráfico do tempo de resposta do teste realizado em ambiente de núvem privada.

4.2.2 Testes com nuvem pública

A abordagem proposta foi avaliada através de dois tipos de testes: testes funcionais, visando comprovar a funcionalidade da abordagem em um ambiente real, e testes de desempenho, para analisar seu comportamento em um ambiente real, sob condições adversas. Para tais testes foi preparado um ambiente com servidores geograficamente separados e instâncias do EC2 da AWS. Tais máquinas se conectavam entre si através da Internet exclusivamente. A Figura 4.3 exibe a topologia do cenário preparado. Cada máquina foi identificada baseada em seu papel dentro do ambiente:

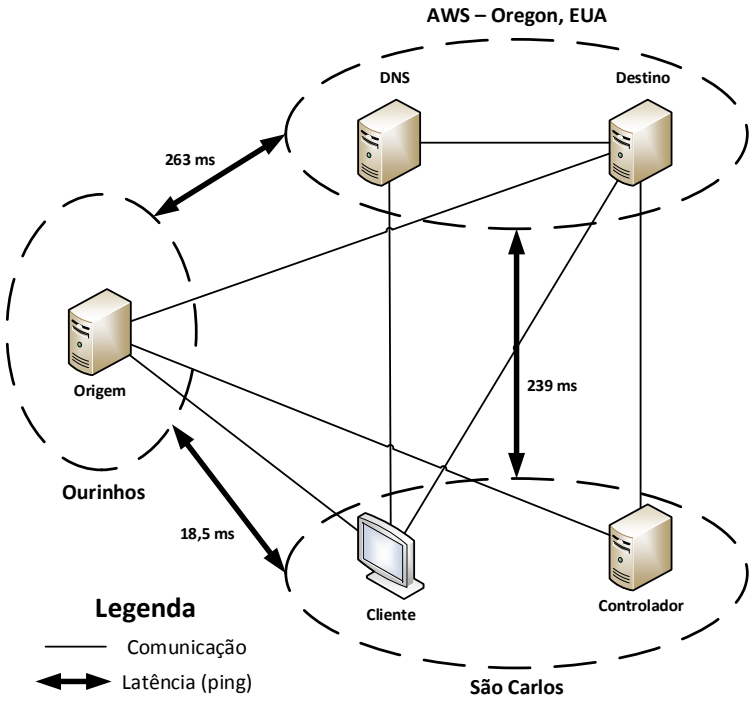


Figura 4.3: Topologia do cenário utilizado nos testes de redirecionamento entre Ourinhos e AWS.

- **Origem**: Executa um serviço *web* cujos fluxos de rede se deseja redirecionar;
- **Destino**: Máquina para onde os fluxos de rede do serviço devem ser redirecionados;
- **Controlador**: Executa o controlador *OpenFlow*;
- **DNS**: Executa o servidor de DNS dinâmico;
- **Cliente**: Executa uma aplicação cliente que faz requisições ao serviço *web* antes, durante e depois do redirecionamento dos fluxos de rede.

4.2.2.1 Configuração de *hardware*

A máquina Origem, localizada em Ourinhos, SP, Brasil, consiste de uma máquina virtual, contendo um processador virtual (VCPU) de 1 núcleo, com 800 MHz de frequência, 1GB de memória RAM e um disco rígido virtual de 20 GB. A máquina Cliente é uma máquina física, localizada em São Carlos, SP, Brasil, com um processador Intel Core i7-870 com 8 núcleos lógicos, cada um com frequência de 2,93 GHz; 16 GB de memória RAM e disco rígido de 1 TB. A máquina Controlador, também localizada em São Carlos, é uma máquina física, com processador Intel Core i5-2400, com 4 núcleos lógicos com 3,10 GHZ cada; 3 GB de RAM e disco de 500 GB.A máquina DNS foi instanciada no EC2 da AWS, sendo do tipo t1.micro, cuja configuração, reportada pela AWS é de 1 VCPU com capacidade de até 2 ECUs [1], 613 MB de RAM e disco rígido virtual de 20 GB.

A máquina Destino também foi instanciada na AWS, porém sua configuração variou conforme o teste executado. A Tabela 4.1 resume os tipos de instância utilizados e suas respectivas configurações. Para todos os testes procurou-se utilizar a mesma *Availability Zone* para todas as instâncias utilizadas: us-west-2a, que se localiza em Boardman, OR, Estados Unidos.

Nome	ECUs	VCPUs (Unidade)	RAM (GiB)	HD (GiB)	Net Perf*
c1.xlarge	20	8	7,00	4×420	Alta
m2.4xlarge	26	8	68,40	2×840	Alta
m3.2xlarge	26	8	30	Somente EBS	Alta

* Performance de rede de cada instância informada pela AWS

Tabela 4.1: Configuração das instâncias utilizadas para Destino

4.2.2.2 Configuração de *software*

Em todas as máquinas do ambiente foi instalado o S.O. *Ubuntu Server* 12.04.3 LTS de 64 bits. A configuração de *software* de Origem e Destino foi idêntica, sendo composta pelo *switch* virtual, um servidor *web*, um cliente do DNS dinâmico e um programa para captura de pacotes. Em Controlador foi instalado o NOS POX, e sobre ele, o controlador desenvolvido para o redirecionamento de fluxos. Em DNS se instalou o servidor DNS dinâmico. Em Cliente foram instalados apenas um programa para captura de pacotes de rede e uma Máquina Virtual Java (JVM) para executar uma aplicação cliente desenvolvida para um dos testes. Além desses foi utilizada a ferramenta *wget*, instalada por padrão no S.O. Foi configurado o endereço de DNS como servidor DNS primário para Origem, Destino e Cliente. A Tabela 4.2 resume os *softwares* instalados e suas respectivas versões.

[1] Na Seção 2.1.2 é descrita a forma como a AWS informa a potência de suas instancias.

Máquina	*Switch* virtual	Servidor *web*	*Software* DNS	Controlador *OpenFlow*	Captura de pacotes
Origem	openvswitch 1.9.0	apache 2.4.6	nsupdate	—	tcpdump
Destino	openvswitch 1.9.0	apache 2.4.6	nsupdate	—	tcpdump
Controlador	—	—	—	pox 1.0.0	—
DNS	—	—	bind	—	—
Cliente	—	—	—	—	tcpdump

Tabela 4.2: *Softwares* instalados nas máquinas do ambiente de testes.

4.2.2.3 Configuração de rede

Todas as máquinas do ambiente foram configuradas com endereços IP públicos, diretamente acessíveis a partir da Internet. Sobre esse aspecto é importante ressaltar a preocupação que se teve em simular um ambiente com controle apenas nas bordas da rede, para demonstrar o funcionamento da abordagem com *OpenFlow* na infraestrutura da Internet atual. No ambiente preparado não havia nenhum controle sobre *firewalls* ou roteadores em nenhum dos sítios. Com as instâncias da AWS, existe certa liberdade de configuração em termos de *firewall*, porém limitada, e monitorada pelo provedor, de forma que comportamentos inseguros são bloqueados. Mesmo assim, as configurações de *firewall* em todos os sítios, inclusive o da AWS, foi a mesma: permitir acesso aos IPs a partir da Internet nas portas reservadas ao servidor *web*, controlador *OpenFlow*, cliente do DNS dinâmico, a conexão com o controlador *OpenFlow*, e acesso remoto, via SSH. Nenhum mecanismo de segurança foi alterado em nenhum dos sítios.

4.2.2.4 Testes de funcionalidade

A princípio foram realizados os testes de funcionalidade, para comprovar que a abordagem desenvolvida se comportava da maneira esperada nesse ambiente. Os testes consistiram no Cliente enviando requisições *web* constantes enquanto se realizava o processo de redirecionamento de fluxos. Para tanto, utilizou-se um *script* simples, baseado na ferramenta *wget*, para fazer requisições GET de um arquivo *index.html* periodicamente (uma requisição por segundo). Para validar o funcionamento da abordagem foram verificados os *logs* dos servidores *web* em Origem e Destino durante todo o processo.

O teste foi realizado duas vezes, primeiramente redirecionando fluxos de Origem para Destino, e depois de Destino para Origem, para garantir que o funcionamento independia de fatores externos ao sistema, como por exemplo a configuração de *firewalls* que poderiam impedir

entrada ou saída de tráfego em algum dos sítios. Em ambos os testes os resultados foram satisfatórios. Todas as requisições realizadas foram atendidas, seja antes, durante ou depois do redirecionamento de fluxos.

4.2.2.5 Testes de desempenho

Para testar o comportamento da arquitetura nesse ambiente, foi simulado um cenário em que um servidor *web* estivesse sobrecarregado e, para contornar o problema, fosse replicado em uma instancia robusta da AWS, e seu tráfego redirecionado para lá, para uma eventual atualização de *hardware*, ou até mesmo para uma migração definitiva. Para que o serviço não deixasse de responder a requisições durante a transição, a arquitetura de redirecionamento de fluxos foi utilizada em conjunto com um mecanismo de migração.

4.2.2.5.1 Geração da carga de trabalho

Para esses testes procurou-se criar condições de acesso realísticas,baseando-se em estudos estatísticos que descrevem o comportamento usual de acessos a servidores *web*. Estudos demonstram que, em geral, os acessos a arquivos em servidores *web* seguem a distribuição de Pareto (MITZENMACHER, 2004), e os tamanhos dos arquivos armazenados nesses servidores seguem a lei de Zipf (GONG, 2001). Sendo assim, na elaboração dos testes procurou-se gerar cargas sintéticas, seguindo essas leis estatísticas.

Em estatística, a distribuição de Pareto é definida por (Equação 4.1)

$$\bar{F}(x) = Pr(X > x) = \begin{cases} (\frac{x_m}{x})^{\alpha} & x \geq x_m \\ 1 & x < x_m \end{cases} \tag{4.1}$$

onde x_m é o valor mínimo (estritamente positivo) possível de X e α é um parâmetro positivo. Sua Função de Distribuição Cumulativa (*Cumulative Distribution Function* - CDF) é dada por (Equação 4.2)

$$F_x = \begin{cases} 1 - (\frac{x_m}{x})^{\alpha} & x \geq x_m \\ 0 & x < x_m \end{cases} \tag{4.2}$$

A distribuição de Zipf é definida por (Equação 4.3)

$$f(k;s,N) = \frac{1/k^s}{\sum_{n=1}^{N}(1/n^s)} \tag{4.3}$$

onde

- N é o número de elementos;
- k é o valor de sua classificação (*rank*);
- s é o valor do expoente caracterizando a distribuição.

A CDF da lei de Zipf é dada por (Equação 4.4)

$$\frac{H_{k,s}}{H_{N,s}} \tag{4.4}$$

Onde $H_{k,s}$ e $H_{N,s}$, são números harmônicos, definidos por (Equações 4.5 e 4.6)

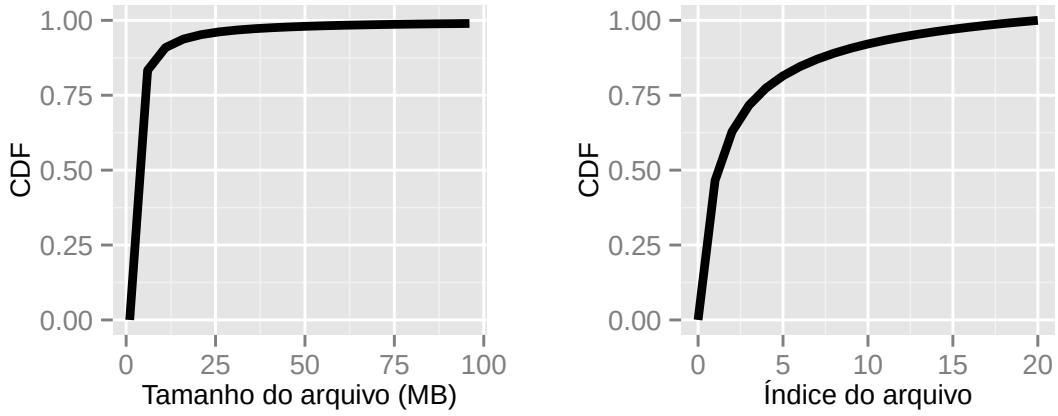
$$H_{k,s} = \sum_{k=1}^{k} \frac{1}{k^s} \tag{4.5}$$

$$H_{N,s} = \sum_{k=1}^{N} \frac{1}{k^s} \tag{4.6}$$

As cargas sintéticas foram geradas da seguinte forma: foram criados 20 arquivos, nomeados 1, 2, ..., 20, cujos tamanhos variaram entre 1 e 100 MB, seguindo a lei de Zipf. O gráfico apresentado na Figura 4.4(a) exibe a CDF dos tamanhos dos arquivos. Através dele observa-se que mais de 75% dos arquivos possui tamanho inferior a 25 MB. A seguir, calculou-se a quantidade de acessos que seriam feitos a cada arquivo, de um total de 200 acessos, seguindo a lei de Pareto. A Figura 4.4(b) mostra a CDF dos números de requisições feitos a cada arquivo. Observa-se aqui que mais de 75% desses acessos devem ser feitos aos arquivos 1 a 10.

4.2.2.5.2 Descrição dos testes

Criadas as cargas sintéticas, iniciou-se os testes. Eles foram executados da seguinte maneira: uma aplicação executando em Cliente enviava rajadas de 200 requisições descritas na Subseção 4.2.2.5.1 a cada 30 segundos. O teste se inicia com as requisições sendo submetidas à Origem. Passados 2 minutos do início das requisições é iniciado o processo de redirecionamento de fluxos. Nos testes foi simulado um tempo de redirecionamento de 6 minutos (durante o redirecionamento). Após esse período as entradas em DNS são atualizadas, e a partir desse ponto, as requisições passam a ser feitas diretamente à Destino (depois do redirecionamento), e



(a) Distribuição do tamanho dos arquivos (b) Distribuição dos acessos a cada arquivo

Figura 4.4: Gráficos de distribuição das cargas sintéticas geradas.

continuam por mais 2 minutos, antes de serem encerradas. Ao todo, requisições são feitas por 10 minutos. Todas as requisições são feitas através de comandos GET, com conexões HTTP não persistentes, ou seja, para cada arquivo requisitado é aberta uma nova conexão. Para cada requisição foram anotados os tempos de início e fim, visando calcular o tempo de resposta de cada uma. Eventuais erros e perdas também foram registrados. Durante o processo todo o tráfego relativo às requisições foi capturado em Origem, Destino e Cliente, através do programa *tcpdump*.

4.2.2.6 Análise dos resultados

A partir dos dados coletados foram feitas as análises a seguir. Primeiramente foram analisados os tempos de resposta para uma instância de cada um dos três tipos. A Figura 4.5 exibe um gráfico de análise de dispersão (média, variância e desvio-padrão) dos tempos de resposta de cada teste, durante todo o processo (antes, durante e depois do redirecionamento). Através deles, observa-se que, das três instâncias testadas, a que obteve menores tempos de resposta durante o processo foi a do tipo *m2.4xlarge*, com uma média de 25 segundos para se obter um arquivo, contra aproximadamente 82 segundos do tipo *m2.8xlarge* e mais de 100 segundos do tipo *c1.xlarge*. Também percebe-se pouca variância e desvio-padrão, o que indicam que os tempos de resposta dessa instância se mantiveram mais constantes ao longo do processo.

Os histogramas da Figura 4.6 mostram a dispersão dos tempos de resposta durante todo o processo. Aravés deles é possível confirmar a maior estabilidade dos tempos de resposta da imagem do tipo *m2.4xlarge*; enquanto as outras duas imagens chegam a ter requisições com tempo total de resposta maiores do que 200 segundos, ela responde a maioria deles em menos

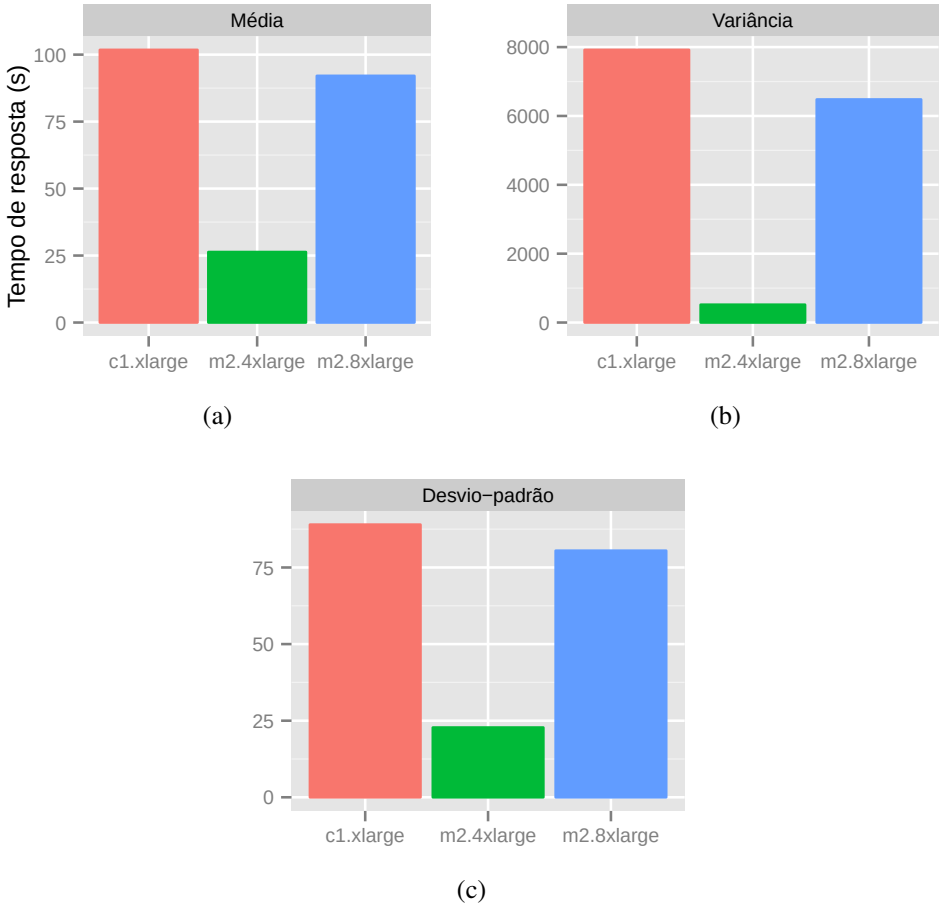


Figura 4.5: Gráficos de análise de dispersão dos tempos de resposta.

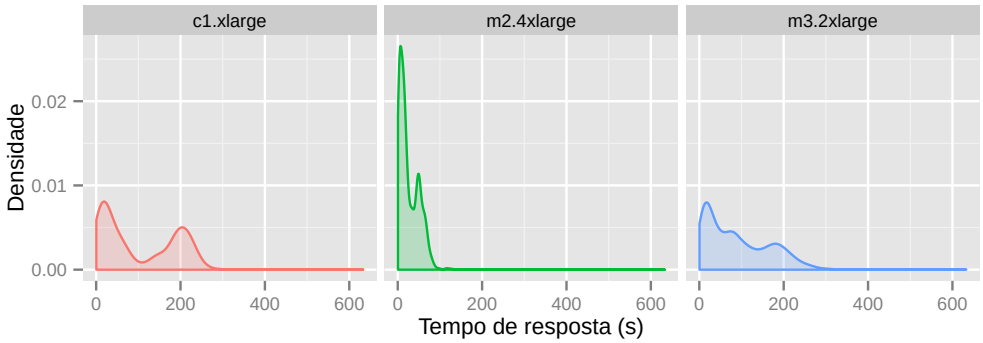


Figura 4.6: Gráficos de distribuição dos tempos de resposta.

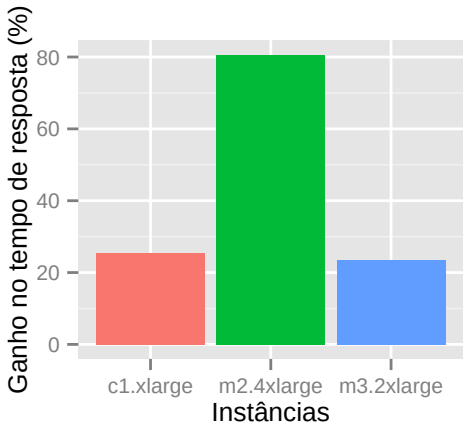


Figura 4.7: Ganho no tempo de resposta comparado com o servidor original.

da metade desse tempo.

O gráfico da Figura 4.7 mostra o ganho no tempo de resposta, calculado com base na comparação entre os tempos de atendimento das requisições antes da migração e depois dela, de acordo com a fórmula (Equação 4.7)

$$1 - (Media_{Antes} / Media_{Depois} * 100) \tag{4.7}$$

Sendo $Media_{Antes}$ o tempo de resposta médio antes da migração e $Media_{Depois}$ o tempo de resposta médio depois da migração. Com base no gráfico pode-se perceber que em todos os casos houve ganho, e que a instância *m2.4xlarge* apresentou um ganho de 80%.

Por fim, o gráfico da Figura 4.8 exibe a porcentagem de requisições não atendidas durante todo o processo. Através deste gráfico é possível perceber que houve uma perda considerável de requisições, mas que no caso da instância *m2.4xlarge*, elas foram menores. Considerando que parte das perdas se deve ao início do processo, onde a máquina na origem estava sobrecarregada e descartando pacotes, pode-se dizer que, ao fim do processo, a migração foi benéfica, pois

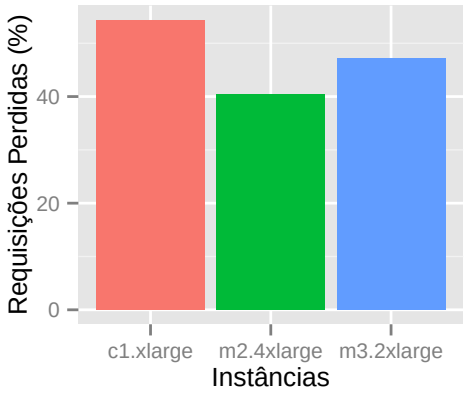


Figura 4.8: Perda de pacotes durante todo o processo.

diminuiu a quantidade de perdas que aconteceriam se toda a carga fosse enviada para a origem apenas. Através do uso da arquitetura proposta isso foi possível sem que a aplicação deixasse de responder.

Dessa forma, podemos concluir que o uso da arquitetura permitiu uma melhora na resposta do serviço em todos os casos, e que no caso da instância *m2.4xlarge*, o ganho no tempo de resposta foi significantemente maior. É interessante ressaltar também que a diferença principal entre as instâncias utilizadas é a quantidade de memória RAM. Todas as instâncias apresentavam o melhor desempenho de rede dentre os tipos de instância oferecidos pelo AWS, o que pode indicar que o fator mais impactante no desempenho foi a memória, e não a rede. Dessa forma, a arquitetura, mesmo aumentando o trajeto dos pacotes, não teve tanto impacto no desempenho final da aplicação.

Capítulo 5

CONCLUSÕES E TRABALHOS FUTUROS

Este capítulo apresenta as principais considerações e conclusões obtidas com o desenvolvimento deste trabalho. Também são apontadas direções para possíveis pesquisas futuras relacionadas a este tema.

5.1 Conclusões

Neste trabalho foi proposta e implementada uma arquitetura para o redirecionamento de fluxos de rede baseada em *OpenFlow* para dar suporte à migração de aplicações entre diferentes provedores de computação em nuvem IaaS. Através de testes em ambientes controlados e testes feitos na Internet, mostrou-se que a arquitetura implementada é aplicável nos casos de serviços disponíveis na Internet e que ela atinge seu objetivo de manter a continuidade de serviço durante a migração de uma aplicação nesse caso.

Foram desenvolvidas duas técnicas para o redirecionamento de fluxos, uma baseada na arquitetura original da Internet e outra baseada na Internet atual, com seus diversos mecanismos de segurança. Os testes da segunda técnica em ambiente real provaram o seu funcionamento e apresentaram resultados razoáveis apesar da estratégia de redirecionamento utilizada não utilizar o menor caminho para o redirecionamento. Dessa forma, acredita-se que caso a primeira técnica seja aplicável, os resultados serão mais satisfatórios.

Através dos testes em ambiente simulado comprovou-se que a primeira técnica pode ser implementada na Internet caso algumas restrições de segurança forem aliviadas. Tendo em vista o crescimento do *OpenFlow* no mercado, acredita-se que no futuro provedores de IaaS comecem a oferecer serviços baseados nessa tecnologia, dando a seus clientes maior controle sobre a rede, o que poderia abrir espaço para a utilização da primeira técnica. Há ainda outros

cenários onde ela poderia ser aplicada, por exemplo, no caso de uma DMZ entre provedores.

Além destes pontos, o desenvolvimento desse trabalho apresenta outra conclusão importante: através dele pôde-se comprovar a hipótese de que é possível se desenvolver uma solução baseada em RDS que funcione na Internet atual sem a necessidade de elementos no núcleo da rede. De fato, através do uso dos *switches OpenFlow* virtuais, provou-se que é possível implementar essa solução sem a necessidade da instalação de nenhum equipamento físico. Isso prova que RDS, e, em especial *OpenFlow* podem oferecer soluções baratas e eficazes mesmo na infraestrutura atual da Internet.

5.2 Trabalhos Futuros

Os testes realizados mostraram a funcionalidade da arquitetura e sua capacidade de redirecionamento de fluxos utilizando a técnica de reescrita de campos de cabeçalhos. Um possível estudo futuro pode comparar essa técnica com as outras técnicas de redirecionamento de fluxos já utilizadas, baseadas em encapsulamento de pacotes, para se medir as vantagens e desvantagens dessa abordagem em relação às soluções atuais.

Os resultados deste trabalho mostraram a viabilidade da arquitetura proposta e que ela é capaz de dar suporte à mecanismos de migração de aplicações. Futuramente, é possível integrá-la a algum mecanismo de sincronização de dados para oferecer um serviço completo de migração de aplicações.

Apendice A

CÓDIGOS-FONTE E *scripts*

Neste Apêndice são listados os trechos do código-fonte do controlador *OpenFlow* implementado que são citados na Seção 3.7.1.1 e o trecho do *script* de configuração do *switch* virtual nas MVs, citado na Seção 3.7.2

Listagem A.1: *Script* de instalação e configuração do *openvswitch* das máquinas virtuais do protótipo funcional implementado.

```
1  #!/bin/bash
2  #variaveis
3  #GW: rota default descoberta atraves do comando ip route
4  #NIC: nome da interface de rede padrao
5  #MAC: endereco mac da interface de rede padrao
6  #(OBS: as vms utilizadas possuem apenas uma interface de rede)
7
8  GW=`ip route|grep default|awk '{print $3}'`
9  NIC=`/sbin/ifconfig | grep -m1 'encap:Ethernet' | cut -d' ' -f1`
10 MAC=`/sbin/ifconfig | grep HWaddr | awk '{print $5}'`
11
12 # fazendo download do openvswitch
13 wget http://openvswitch.org/releases/openvswitch-1.9.0.tar.gz
14 #compilando
15 tar xzf openvswitch-1.9.0.tar.gz
16 cd /root/openvswitch-1.9.0
17 ./configure --with-linux=/lib/modules/`uname -r`/build
18 make && make install
19 #ativando o modulo
20 insmod datapath/linux/openvswitch.ko
21 make modules_install
22
```

```
23 #gerando configuracao inicial do openvswitch
24 ovsdb-tool create /usr/local/etc/openvswitch/conf.db vswitchd/vswitch.
       ovsschema
25
26 #iniciando ovsdb-server
27 if ! ps -ef | grep -v grep | grep ovsdb-server > /dev/null; then
28   ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock \
29                --remote=db:Open_vSwitch,manager_options \
30                --private-key=db:SSL,private_key \
31                --certificate=db:SSL,certificate \
32                --bootstrap-ca-cert=db:SSL,ca_cert \
33                --pidfile --detach
34 fi
35 ovs-vsctl --no-wait init
36
37 #iniciando ovs-vswitchd
38 if ! ps ef | grep -v grep | grep ovswitchd > /dev/null; then
39   ovs-vswitchd --pidfile --log-file --detach
40 fi
41
42 #criando uma bridge no linux atraves do openvswitch
43 #que funcionara como switch virtual e sera controlada via OpenFlow
44 if ! ovs-vsctl show | grep br0 > /dev/null; then
45   ovs-vsctl add-br br0
46   ovs-vsctl set bridge br0 other-config:hwaddr=$MAC
47   ovs-vsctl add-port br0 $NIC
48 fi
49
50 #reconfigurando a interface de rede
51 #para funcionar com o openvswitch e a configurando
52 #como gateway padrao da maquina virtual
53 ifconfig $NIC 0
54 dhclient br0
55 if ip route show | grep default | grep $NIC; then
56   ip route del default via $GW dev $NIC
57 fi
```

A seguir são listados trechos do código-fonte do controlador *OpenFlow* implementado. Para melhor entendimento do mesmo, seguem alguns detalhes de sua implementação:

- O controlador assume que os *switches* que se conectarão a ele possuem apenas duas portas: uma ligada à MV e uma ligada ao *gateway*, da forma ilustrada na Seção 3.3.1;
- As constantes *SWGWPORT* e *SWHOSTPORT* representam, respectivamente, à porta do *switch* ligada ao *gateway* e à porta ligada à MV;
- Como foi implementado apenas para fins de prova de conceito, este protótipo trata apenas de pacotes IPv4 e ARP.

Listagem A.2: Função que trata dos pacotes antes do momento de redirecionamento de fluxos

```
"""
Estrategia padrao de encaminhamento de pacotes:
Pacotes que vem da porta ligada ao host sao encaminhados para o gateway
e vice-e-versa.
"""
def handle_packets_default (self, event):
treated = [ethernet.IP_TYPE, ethernet.ARP_TYPE]
if event.parsed.type in treated:
  log.debug("pt: %s , pkt: %s", event.port, event.parsed)
  if self.macgate is None:
    log.debug("aprendendo mac gateway...")
    self.macgate = event.parsed.src
  rule = of.ofp_flow_mod()
  rule.match = of.ofp_match.from_packet(event.parsed)
  rule.idle_timeout = ITMOUT
  rule.hard_timeout = HTMOUT
  rule.buffer_id = event.ofp.buffer_id

  if event.port == SWGWPORT:
    # se for arp response de outra subrede
    # e do gateway, entao guarda ip e mac
    if isinstance(event.parsed.payload, arp):
      a = event.parsed.payload
      if a.opcode == arp.REPLY:
        log.debug("ARP REPLY")
        self.macgate = a.hwsrc
        self.ipgate = a.protosrc
        log.debug("aprendido mac gateway: %s", self.macgate)

```

```
30      rule.actions.append(of.ofp_action_output(port=SWHOSTPORT))
31    elif event.port == SWHOSTPORT:
32      # se for arp request, guarda o ip e o mac do host
33      if isinstance(event.parsed.payload, arp):
34        a = event.parsed.payload
35        if a.opcode == arp.REQUEST:
36          log.debug("ARP REQUEST")
37          self.machost = a.hwsrc
38          self.iphost = a.protosrc
39          log.debug("aprendido: iphost %s", self.iphost)
40      rule.actions.append(of.ofp_action_output(port=SWGWPORT))
41
42    event.connection.send(rule)
43  else:
44    log.debug("Pacote nao tratado")
```

Listagem A.3: Função que trata dos fluxos a partir do início do redirecionamento de fluxos.

```
def handle_packets_mig (self, event):
  self.refused = 1
  pacote = event.parsed
  treated = [ethernet.IP_TYPE, ethernet.ARP_TYPE]
  if event.parsed.type in treated:
    ### PACOTES DO HOST PARA O GATEWAY ###
    if event.port == SWHOSTPORT:
      log.debug("pacote saindo do host, encaminhando")
      self.forwardTo(event,SWGWPORT)

    ### PACOTES DO GATEWAY PARA O HOST ###
    elif event.port == SWGWPORT:
      log.debug("pacote chegando no host")
      # tratando pacotes ARP
      if isinstance(pacote.payload, arp):
        pktarp = pacote.payload
        if pktarp.opcode == arp.REPLY:
          # ARP REPLY DO GATEWAY PARA O HOST
          log.debug("Chegou um arp reply no switch vindo do gateway.
              Encaminhando p host")
          self.forwardTo(event,SWHOSTPORT)
        elif pktarp.opcode == arp.REQUEST:
          # ARP REQUEST DO GATEWAY PARA O HOST
          if pktarp.protodst == self.iphost:
            log.debug("ARP request chegando do gateway. Respondendo...")
             self.answerARPrequest(event, of.OFPP_IN_PORT)

      # tratando pacotes IPV4
      elif isinstance(pacote.payload, ipv4):
        ip = pacote.payload
        # so tratar os pacotes direcionados ao host
        if ip.dstip == self.iphost:
          if isinstance(ip.payload, tcp) or isinstance(ip.payload, udp):
            # PACOTE TCP/UDP DO GATEWAY PARA O HOST
            # se o pacote nao vem do destino, eh cliente, entao
            # redireciona para o srv destino
            tcpudp = ip.payload
            if tcpudp.dstport in self.open_ports:
              self.forwardTo(event,SWHOSTPORT)
            else:
              if ip.srcip != self.ip_target:
```

```
41                  log.debug("pacote nao vem do destino, deve ser d cliente")
42                  #checando se esse fluxo ja existe na tabela
43                  flux = (ip.srcip,tcpudp.srcport)
44                  if flux in self.vai:
45                    log.debug("ja estava na tabela")
46                    self.redirectTo(event,self.iphost,self.vai[flux],self.
                          ip_target,tcpudp.dstport,of.OFPP_IN_PORT)
47                    self.refused = 0
48                  else:
49                    log.debug("n estava na tabela, inserindo")
50                    # checando se existe um indice de porta disponivel
51                    if self.portindex:
52                      #alocando uma nova porta para uma nova linha na tabela
53                      newport = self.portindex.pop()
54                      if newport not in self.vai and flux not in self.volta:
55                        # adicionando nova entrada na tabela de fluxos
                              redirecionados
56                        self.vai[flux] = newport
57                        self.volta[newport] = flux
58                        log.debug("Tabela: %s", self.vai)
59                        # redirecionando fluxo
60                        log.debug("Redirecionando o pacote: %s,%s,%s,%s",self.
                              iphost,newport,self.ip_target,tcpudp.dstport)
61                        self.redirectTo(event,self.iphost,newport, self.
                              ip_target, tcpudp.dstport, of.OFPP_IN_PORT)
62                        self.refused = 0
63                      else:
64                        log.error("As tabelas tao com fluxos q nao deveriam
                              estar nelas. Verificar.")
65                        self.redirectTo(event,self.iphost,newport, self.
                              ip_target, tcpudp.dstport, of.OFPP_IN_PORT)
66                        self.refused = 0
67                      else:
68                        log.error("numero de portas disponiveis esgotado")
69                         # ver o que fazer se estourar o portindex
70                         pass
71                else:
72                  # se o pacote vem do destino, deve ser redirecionado de volta
                        para o cliente
73                  log.debug("pacote voltando do destino. Mandando pro cliente")
74                  p = tcpudp.dstport
75                  if p in self.volta:
```

```
76                  log.debug("Redirecionando para: (%s,%s,%s,%s)",self.iphost,
                        tcpudp.srcport,self.volta[p][0],self.volta[p][1])
77                  self.redirectTo(event,self.iphost,tcpudp.srcport,self.volta
                        [p][0],self.volta[p][1],of.OFPP_IN_PORT)
78                  self.refused = 0
79                elif len(search2) == 0:
80                  log.error("nao achou a porta na tabela.")
81                  pass
82
83              if self.refused == 1:
84                log.error("Pacote nao tratado")
85          else:
86            #OUTROS PACOTES IPV4 DO GATEWAY PARA O HOST
87            self.forwardTo(event,SWHOSTPORT)
88  else:
89    log.debug("Pacote nao tratado")
```

Listagem A.4: Função que cria regra para encaminhar fluxo para determinada porta do *switch*.

```
def forwardTo(self, event, swpt):
  rule = of.ofp_flow_mod()
  rule.match = of.ofp_match.from_packet(event.parsed)
  log.debug("ForwardTo match: (%s, %s, %s, %s)", rule.match.nw_src, rule.
     match.tp_src, rule.match.nw_dst, rule.match.tp_dst)
  rule.idle_timeout = ITMOUT
  rule.hard_timeout = HTMOUT
  rule.buffer_id = event.ofp.buffer_id
  rule.actions.append(of.ofp_action_output(port=swpt))
  event.connection.send(rule)
```

Listagem A.5: Função que cria regra para redirecionar fluxo para a MV no sítio destino

```
def redirectTo(self, event, ipsrc, ptsrc, ipdst, ptdst, swpt):
  rule = of. ofp_flow_mod()
  rule.flags = of.OFPFF_SEND_FLOW_REM
  rule.match = of.ofp_match.from_packet(event.parsed)
  log.debug("RedirectTo match: (%s, %s, %s, %s)", rule.match.nw_src, rule.
     match.tp_src, rule.match.nw_dst, rule.match.tp_dst)
  rule.idle_timeout = ITMOUT
  rule.hard_timeout = HTMOUT
  rule.buffer_id = event.ofp.buffer_id
  acts = []
  acts.append(of.ofp_action_dl_addr(type=of.OFPAT_SET_DL_DST, dl_addr=self.
     macgate))
  acts.append(of.ofp_action_dl_addr(type=of.OFPAT_SET_DL_SRC, dl_addr=self.
     machost))
  acts.append(of.ofp_action_nw_addr(type=of.OFPAT_SET_NW_SRC, nw_addr=ipsrc))
  acts.append(of.ofp_action_nw_addr(type=of.OFPAT_SET_NW_DST, nw_addr=ipdst))
  acts.append(of.ofp_action_tp_port(type=of.OFPAT_SET_TP_SRC, tp_port=ptsrc))
  acts.append(of.ofp_action_tp_port(type=of.OFPAT_SET_TP_DST, tp_port=ptdst))
  acts.append(of.ofp_action_output(port=swpt))
  rule.actions = acts
  event.connection.send(rule)
```

REFERÊNCIAS

Amazon.com, Inc. *Amazon Elastic Block Store (EBS)*. 2014. Disponível em: <http://aws.amazon.com/pt/ebs/>. Acesso em: 07 fev. 2014.

Amazon.com, Inc. *Amazon Elastic Compute Cloud (EC2)*. 2014. Disponível em: <http://aws.amazon.com/ec2>. Acesso em: 28 jan. 2014.

Amazon.com, Inc. *Amazon Web Services*. 2014. Disponível em: <http://aws.amazon.com>. Acesso em: 28 jan. 2014.

ANDERSON, T. et al. Overcoming the internet impasse through virtualization. *Computer*, Los Alamitos, v. 38, n. 4, p. 34–41, apr. 2005. ISSN 0018-9162. Disponível em: <http://dx.doi.org/10.1109/MC.2005.136>.

AppZero. *Enterprise Application Migration*. 2013. Disponível em: <http://www.appzero.com/>. Acesso em: 08 jan. 2014.

AYMERICH, F.; FENU, G.; SURCIS, S. In: FIRST INTERNATIONAL CONFERENCE ON THE APPLICATIONS OF DIGITAL INFORMATION AND WEB TECHNOLOGIES (ICADIWT '08). *Proceedings...* [S.l.].

BANERJEE, J. Moving to the cloud: Workload migration techniques and approaches. In: 19TH INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE COMPUTING (HiPC '12). *Proceedings...* [S.l.]: IEEE, 2012. p. 1–6.

BARHAM, P. et al. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, New York, v. 37, n. 5, p. 164–177, oct. 2003. ISSN 0163-5980. Disponível em: <http://doi.acm.org/10.1145/1165389.945462>.

BOUGHZALA, B. et al. OpenFlow supporting inter-domain virtual machine migration. In: EIGHTH INTERNATIONAL CONFERENCE ON WIRELESS AND OPTICAL COMMUNICATIONS NETWORKS (WOCN '11). *Proceedings...* [S.l.], 2011. p. 1–7.

BRADFORD, R. et al. In: *Proceedings...* [S.l.: s.n.].

BUYYA, R.; RANJAN, R.; CALHEIROS, R. N. In: *Proceedings...* [S.l.: s.n.].

CHAUHAN, M.; BABAR, M. Towards process support for migrating applications to cloud computing. In: 2012 INTERNATIONAL CONFERENCE ON CLOUD AND SERVICE COMPUTING (CSC '12). *Proceedings...* [S.l.]: IEEE, 2012. p. 80–87.

CLARK, C. et al. Live migration of virtual machines. In: 2ND CONFERENCE ON SYMPOSIUM ON NETWORKED SYSTEMS DESIGN & IMPLEMENTATION (NSDI '05). *Proceedings...* USENIX Association, 2005. p. 273–286. Disponível em: <http://dl.acm.org/citation.cfm?id=1251203.1251223>.

DIKAIAKOS, M. et al. Cloud computing: Distributed internet computing for IT and scientific research. *Internet Computing, IEEE*, Los Alamitos, v. 13, n. 5, p. 10–13, set. 2009. ISSN 1089-7801.

EHRENKRANZ, T.; LI, J. On the state of ip spoofing defense. *ACM Transactions on Internet Technology*, ACM, New York, v. 9, n. 2, p. 6:1–6:29, may 2009. ISSN 1533-5399. Disponível em: <http://doi.acm.org/10.1145/1516539.1516541>.

Eucalyptus Systems, Inc. *Eucalyptus: Open Source AWS Compatible Private Clouds*. 2014. Disponível em: <https://www.eucalyptus.com/>.

GARFINKEL, S.; ABELSON, H. *Architects of the Information Society: 35 Years of the Laboratory for Computer Science at MIT*. 1 ed. Cambridge: MIT Press, 1999. 72 p. ISBN 9780262071963. Disponível em: <http://books.google.com.br/books?id=Fc7dkLGLKrcC>.

GONG, W. et al. On the tails of web file size distributions. In: 39TH ALLERTON CONFERENCE ON COMMUNICATION, CONTROL, AND COMPUTING. *Proceedings...* [S.l.], 2001.

GUDE, N. et al. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, New York, v. 38, n. 3, p. 105–110, jul. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1384609.1384625>.

HADAS, D.; GUENENDER, S.; ROCHWERGER, B. *Virtual network services for federated cloud computing*. [S.l.], nov 2009.

HAND, S. et al. In: . [S.l.: s.n.].

HAO, F. et al. Enhancing dynamic cloud-based services using network virtualization. *ACM SIGCOMM Computer Communication Review*, New York, v. 40, n. 1, p. 67–74, jan. 2010. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1672308.1672322>.

HIROFUCHI, T. et al. A live storage migration mechanism over WAN for relocatable virtual machine services on clouds. In: 9TH INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID (CCGRID '09). *Proceedings...* IEEE, 2009. p. 460–465. ISBN 978-0-7695-3622-4. Disponível em: <http://dx.doi.org/10.1109/CCGRID.2009.44>.

HIROFUCHI, T. et al. A multi-site virtual cluster system for wide area networks. In: FIRST USENIX WORKSHOP ON LARGE-SCALE COMPUTING (LASCO '08). *Proceedings...* USENIX Association, 2008. p. 4:1–4:10. Disponível em: <http://dl.acm.org/citation.cfm?id=1411725.1411729>.

LEVY, S. *Going With the Flow: Google's Secret Switch to the Next Wave of Networking*. 2010. Disponível em: <http://www.wired.com/wiredenterprise/2012/04/going-with-the-flow-google/>. Acesso em: 08 jan. 2014.

McKeown, N. et al. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, New York, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>.

MELL, P.; GRANCE, T. *The NIST Definition of Cloud Computing*. Gaithersburg, MD, set. 2011. Disponível em: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>.

MITZENMACHER, M. Dynamic models for file sizes and double pareto distributions. *Internet Mathematics*, Natick, v. 1, n. 3, p. 305–333, 2004.

NAGIN, K. et al. Inter-cloud mobility of virtual machines. In: 4TH ANNUAL INTERNATIONAL CONFERENCE ON SYSTEMS AND STORAGE (SYSTOR '11). *Proceedings...* ACM, 2011. p. 3:1–3:12. ISBN 978-1-4503-0773-4. Disponível em: <http://doi.acm.org/10.1145/1987816.1987820>.

NASCIMENTO, M. R. et al. Virtual routers as a service: The routeflow approach leveraging software-defined networks. In: 6TH INTERNATIONAL CONFERENCE ON FUTURE INTERNET TECHNOLOGIES (CFI '11). *Proceedings...* [S.l.].

NETIQ CORPORATION. *PlateSpin Migrate*. 2014. Disponível em: <https://www.netiq.com/products/migrate/>. Acesso em: 08 jan. 2014.

NICIRA, INC. *About POX*. 2012. Disponível em: <http://www.noxrepo.org/pox/about-pox/>.

No-IP.com. *Free Dynamic DNS - Managed DNS - Managed Email - Domain Registration - No-IP*. 2014. Disponível em: <http://www.noip.com/>.

NUNES, R. V.; PONTES, R. L.; GUEDES, D. In: *Anais...* [S.l.: s.n.].

OPEN NETWORKING FOUNDATION. *The OpenFlow Switch Specification*. 2011. Disponível em: <http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf>.

Open Networking Foundation. *Software-Defined Networking: The New Norm for Networks*. Palo Alto, CA, USA, abr. 2012. Disponível em: <http://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>.

PARKHILL, D. F. *The Challenge of the Computer Utility*. London: Addison-Wesley Publishing Company, 1966. 207 p.

PFAFF, B. et al. Extending networking into the virtualization layer. In: 8TH WORKSHOP ON HOT TOPICS IN NETWORKS (HotNets-VIII). *Proceedings...* [S.l.]: ACM, 2009. p. 1–6.

SHALOM, N. *Mapping the Cloud/PaaS Stack*. 2012. Disponível em: <http://natishalom.typepad.com/nati_shaloms_blog/2012/05/mapping-the-cloudpaas-stack.html>.

TRAVOSTINO, F. et al. Seamless live migration of virtual machines over the MAN/WAN. *Future Generation Computer Systems*, Amsterdan, v. 22, n. 8, p. 901–907, oct. 2006. ISSN 0167-739X. Disponível em: <http://dx.doi.org/10.1016/j.future.2006.03.007>.

TRIDGELL, A. *Efficient Algorithms for Sorting and Synchronization*. Tese (Doutorado) — Australian National University, 1999.

TURNER, J.; TAYLOR, D. Diversifying the internet. In: 2005 GLOBAL TELECOMMUNICATIONS CONFERENCE (GLOBECOM '05). *Proceedings...* [S.l.].

VIXIE, P. et al. *Dynamic Updates in the Domain Name System (DNS UPDATE)*. IETF, abr. 1997. RFC 2136 (Proposed Standard). (Request for Comments, 2136). Updated by RFCs 3007, 4035, 4033, 4034. Disponível em: <http://www.ietf.org/rfc/rfc2136.txt>.

GLOSSÁRIO

ARP – Address Resolution Protocol

ASP – Application Service Providers

AWS – Amazon Web Services

DDOS – Distributed Denial of Service

DHCP – Dynamic Host Control Protocol

DMZ – DeMilitarized Zone

DNS – Domain Name Service

EC2 – Elastic Compute Cloud

ECU – EC2 Compute Unit

IaaS – Infrastructure as a Service

IP – Internet Protocol

IPv6 – Internet Protocol version 6

JVM – Java Virtual Machine

MAC – Media Access Control

MV – Máquina Virtual

NAT – Network Address Translation

NOS – Network Operating System

OSI – Open Systems Interconnection

OSPF – Open Shortest Path First

OVS – Open vSwitch

QoS – Quality of Service

RDS – Rede Definida Por Software

SO – Sistema Operacional

SSH – Secure Shell

TCAM – Ternary Content Access Memory

TCP – Transport Control Protocol

TIC – Tecnologia da Informação e Comunicação

UDP – User Datagram Protocol

VLAN – Virtual Local Area Network

VPN – Virtual Private Network

WAN – Wide Area Network