Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Engenharia de Computação e Automação Industrial
Área de concentração: Engenharia de Computação

Proposta e Validação de Nova Arquitetura de Redes de Data Center

Autor: Carlos Alberto Braz Macapuna
Orientador: Maurício Ferreira Magalhães
Co-orientador: Christian Esteve Rothenberg

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Campinas, SP
2011

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Macapuna, Carlos Alberto Braz
M118p Proposta e validação de nova arquitetura de redes de data center / Carlos Alberto Braz Macapuna. – Campinas, SP: [s.n.], 2011.

Orientadores: Maurício Ferreira Magalhães; Christian Esteve Rothenberg.
Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes de computadores. 2. Arquitetura de redes de computadores). 3. Centro de processamento de dados. 4. Redes de computadores (Gerenciamento). 5. Redes de computadores - Protocolos. I. Magalhães, Maurício Ferreira. II. Rothenberg, Christian Esteve. III. Universidade Estadual de Campinas. . Faculdade de Engenharia Elétrica e de Computação. IV. Título

Título em Inglês:	Proposal and Validation of New Architecture for Data Center Networks
Palavras-chave em Inglês:	Computer networks, Architecture of computer networks, Data processing center, Computer networks (Management), Computer networks - Protocols
Área de concentração:	Engenharia de Computação
Titulação:	Mestre em Engenharia Elétrica
Banca Examinadora:	Edmundo Roberto Mauro Madeira, Marcos Rogério Salvador
Data da defesa:	29/04/2011
Programa de Pós Graduação:	Engenharia Elétrica

Carlos Alberto Braz Macapuna

Proposta e Validação de Nova Arquitetura de Redes de Data Center

Tese de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.
Aprovação em 29/04/2011

Banca Examinadora:
Prof. Dr. Maurício Ferreira Magalhães - UNICAMP
Prof. Dr. Edmundo Roberto Mauro Madeira - UNICAMP
Dr. Marcos Rogério Salvador - CPqD

Campinas, SP
2011

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Carlos Alberto Bráz Macapuna

Data da Defesa: 29 de abril de 2011

Título da Tese: "Proposta e Validação de Nova Arquitetura de Redes de Data Center"

Prof. Dr. Maurício Ferreira Magalhães (Presidente): _____

Dr. Marcos Rogério Salvador: _____

Prof. Dr. Edmundo Roberto Mauro Madeira: _____

Resumo

Assim como as grades computacionais, os centros de dados em nuvem são estruturas de processamento de informações com requisitos de rede bastante exigentes. Esta dissertação contribui para os esforços em redesenhar a arquitetura de centro de dados de próxima geração, propondo um serviço eficaz de encaminhamento de pacotes, que explora a disponibilidade de *switches* programáveis com base na API OpenFlow. Desta forma, a dissertação descreve e avalia experimentalmente uma nova arquitetura de redes de centro de dados que implementa dois serviços distribuídos e resilientes a falhas que fornecem as informações de diretório e topologia necessárias para codificar aleatoriamente rotas na origem usando filtros de Bloom no cabeçalho dos pacotes. Ao implantar um exército de gerenciadores de Rack atuando como controladores OpenFlow, a arquitetura proposta denominada *Switching with in-packet Bloom filters* (SiBF) promete escalabilidade, desempenho e tolerância a falhas. O trabalho ainda defende a ideia que o encaminhamento de pacotes pode tornar-se um serviço interno na nuvem e que a sua implementação pode aproveitar as melhores práticas das aplicações em nuvem como, por exemplo, os sistemas de armazenamento distribuído do tipo par <chave,valor>. Além disso, contrapõe-se ao argumento de que o modelo de controle centralizado de redes (*OpenFlow*) está vinculado a um único ponto de falhas. Isto é obtido através da proposta de uma arquitetura de controle fisicamente distribuída, mas baseada em uma visão centralizada da rede resultando, desta forma, em uma abordagem de controle de rede intermediária, entre totalmente distribuída e centralizada.

Palavras-chave: Centro de Dados, Filtro de Bloom, Redes Distribuídas, Computação na Nuvem.

Abstract

Cloud data centers, like computational Grids, are information processing fabrics with very demanding networking requirements. This work contributes to the efforts in re-architecting next generation data centers by proposing an effective packet forwarding service that exploits the availability of programmable switches based on the OpenFlow API. Thus, the dissertation describes and experimentally evaluates a new architecture for data center networks that implements two distributed and fault-tolerant services that provide the directory and topology information required to encode randomized source routes with in-packet Bloom filters. By deploying an army of Rack Managers acting as OpenFlow controllers, the proposed architecture called Switching with in-packet Bloom filters (SiBF) promises scalability, performance and fault-tolerance. The work also shows that packet forwarding itself may become a cloud internal service implemented by leveraging cloud application best practices such as distributed key-value storage systems. Moreover, the work contributes to demystify the argument that the centralized controller model of OpenFlow networks is prone to a single point of failure and shows that direct network controllers can be physically distributed, yielding thereby an intermediate approach to networking between fully distributed and centralized.

Keywords: Computer Networks, Datacenter, Internet, Virtualization, Bloom Filter, Distributed Networks, Cloud Computing.

Agradecimentos

Ao meu orientador Prof. Dr. Maurício Ferreira Magalhães e co-orientador Dr. Christian Esteve Rothenberg, sou grato pela orientação, contribuição e empenho que dedicaram para que esta dissertação se tornasse possível.

Aos amigos de Campinas e do Laboratório de Computação e Automação (LCA), Claudenicio Ferreira, Daniel Moraes, Fábio Verdi, Jáder Moura, Luciano de Paula, Rafael Pasquini, Rodolfo Villaça e Walter Wong, pelas críticas e sugestões, além da amizade e apoio que recebi durante a estadia em Campinas.

À Mirian Rose, pela dedicação, carinho e companheirismo durante esta jornada.

À minha família e amigos de Belém, que sempre estiveram ao meu lado, dando forças e incentivos.

À Unicamp pelo ensino de qualidade e, à CAPES e Ericsson, pelo apoio financeiro.

À minha mãe, Maria José Braz Macapuna.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Acrônimos

AA . . . . . *Application Address*

AMAC . . . . . *Actual MAC*

API . . . . . *Application Programming Interface*

AR . . . . . *Access Router*

ARP . . . . . *Address Resolution Protocol*

ARPANET . . . . . *Advanced Research and Projects Agency Network*

BGP . . . . . *Border Gateway Protocol*

BR . . . . . *Border Router*

D-ITG . . . . . *Distributed Internet Traffic Generator*

DCN . . . . . *Data Center Networks*

DHCP . . . . . *Dynamic Host Configuration Protocol*

DIP . . . . . *Direct IP*

DlBF . . . . . *Deletable Bloom filter*

DNS . . . . . *Domain Name System*

DoD . . . . . *Department of Defense*

DS . . . . . *Directory Service*

ECMP . . . . . *Equal Cost MultiPath*

FPGA . . . . . *Field-programmable Gate Array*

FQDN . . . . . *Fully Qualified Domain Name*

Hi3 . . . . . *Host Identity Indirection Infrastructure*

iBF . . . . . *in-packet Bloom filter*

IP . . . . . *Internet Protocol*

LA . . . . . *Locator Address*

LAN . . . . . *Local Area Network*

LB . . . . . *Load Balancer*

LDM . . . . . *Location Discovery Message*

LDP . . . . . *Location Discovery Protocol*

LLDP . . . . . *Link Layer Discovery Protocol*

MAC . . . . . *Media Access Control*

MPLS . . . . . *Multi Protocol Label Switching*

NAT . . . . . *Network Address Translation*

NoSQL . . . . . *Not only SQL*

NSF . . . . . *National Science Foundation*

OpenFlowVMS . . . . . *OpenFlow Virtual Machine Simulation*

Pcap . . . . . *Packet capture*

PMAC . . . . . *Pseudo MAC*

RM . . . . . *Rack Manager*

RMC . . . . . *Rack Manager Core*

ROFL . . . . . *Routing on Flat Labels*

RSM . . . . . *Replicated State Machine*

RTT . . . . . *round-trip delay time*

SiBF . . . . . *Switching with in-packet Bloom filters*

SSH . . . . . *Secure Shell*

SSL . . . . . *Secure Socket Layer*

STP . . . . . *Spanning Tree Protocol*

TCP . . . . . *Transmission Control Protocol*

TI . . . . . Tecnologia da Informação

TLV . . . . . *Type-length-value*

TM . . . . . *Traffic Matrix*

ToR . . . . . *Top of Rack*

TS . . . . . *Topology Service*

UDP . . . . . *User Datagram Protocol*

VDE . . . . . *Virtual Distributed Ethernet*

VIP . . . . . *Virtual IP*

VL2 . . . . . *Virtual Layer 2*

VLAN . . . . . *Virtual Local Area Network*

VLB . . . . . *Valiant Load Balancing*

VM . . . . . *Virtual Machine*

VPN . . . . . *Virtual Private Network*

WAN . . . . . *Wide Area Network*

WWW . . . . . *World Wide Web*

Trabalhos Publicados Pelo Autor

1. Carlos A. B. Macapuna, Christian E. Rothenberg, Maurício F. Magalhães. "In-packet Bloom filter based data center networking with distributed OpenFlow controllers". *IEEE International Workshop on Management of Emerging Networks and Services (IEEE MENS 2010)* in conjunction with IEEE GLOBECOM 2010, Miami, Florida, USA, 6-10 de Dezembro de 2010.

2. Carlos A. B. Macapuna, Christian E. Rothenberg, Maurício F. Magalhães. "Controle distribuído em Redes de Data Center baseado em Gerenciadores de Rack e Serviços de Diretório/Topologia". *VIII Workshop em Clouds, Grids e Aplicações* (WCGA 2010), Gramado, Rio Grande do Sul, Brasil, 28 de Maio, 2010.

3. Christian E. Rothenberg, Carlos A. B. Macapuna, Fábio L. Verdi, Maurício F. Magalhães, Alexander Zahemszky. "Data center networking with in-packet Bloom filters". *XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos* (SBRC 2010), Gramado, Rio Grande do Sul, Brasil, 24 a 28 de Maio, 2010.

4. Christian E. Rothenberg, Carlos. A. B. Macapuna, Fábio L. Verdi and Maurício F. Magalhães. "The Deletable Bloom filter: A new member of the Bloom family". *IEEE Communications Letters*, vol.14, no.6, pp.557-559, June 2010.

5. Christian E. Rothenberg, Carlos A. B. Macapuna, Maurício Ferreira Magalhães, Fábio L. Verdi, Alexander Wiesmaier, "In-packet Bloom filters: Design and networking applications", *Computer Networks*, In Press, Corrected Proof, Available online 19 December 2010, ISSN 1389-1286.

Capítulo 1

Introdução

Em meio ao cenário de Guerra Fria da década de 50 e 60, a atual Internet começou a dar seus primeiros passos. Surgiu a partir de pesquisas militares com objetivos de proteção militar dos Estados Unidos. O principal objetivo era a manutenção do serviço, mesmo que ocorresse um ataque inimigo a algum ponto crítico da rede ou dos meios de comunicação tradicionais, já que as comunicações militares naquela época passavam pela rede pública de telefonia. A ideia inicial era garantir que a rede continuasse funcionando mesmo com um ataque nuclear, por exemplo. A então *Advanced Research and Projects Agency Network* (ARPANET), a precursora da Internet, cuja implementação inicial consistia de minicomputadores conectados por linhas de transmissão de 56 kbps, também começou a ser utilizada no meio acadêmico, por professores e alunos das universidades. A partir da década de 70, com a diminuição da tensão na Guerra Fria e a crescente utilização da rede para fins não militares, a ARPANET começou a sua fase de expansão, principalmente pela comunidade acadêmica. Foi nessa década que surgiu o modelo TCP/IP, criado para manipular conexões sobre inter-redes. No final da década de 70, foi desenvolvido a rede da *National Science Foundation* (NSF), a NSFNET, devido a ARPANET ser limitada às universidades que tinham um contrato de pesquisa com o Departamento de Defesa dos Estados Unidos (*Department of Defense* (DoD)). A NSFNET cresceu rapidamente, atingindo outros segmentos como bibliotecas e museus. Esse contínuo crescimento levou a NSFNET a perceber que o governo não podia continuar financiando a rede, e logo empresas entraram no ramo. Na década de 80, com novas redes sendo conectadas à ARPANET, localizar os *hosts* estava ficando cada vez mais dispendioso, com isso, surgiu o *Domain Name System* (DNS) que, aliado ao modelo TCP/IP e à interconexão da NSFNET com a ARPANET, originaram o que é conhecido hoje como Internet.

Somente na década de 90, com o surgimento da *World Wide Web* (WWW) e a possibilidade de criação de páginas gráficas para navegação, a Internet tornou-se popular a níveis globais jamais imaginados à época de sua criação. Já na década de 2000, as redes sociais iniciaram uma nova era

da Internet. Hoje, com a complexidade de tecnologias, serviços e finalidades, a Internet já apresenta muitos gargalos, como a escassez dos endereços do protocolo de interconexão (*Internet Protocol* (IP)) e o aumento na tabela de rotas do *Border Gateway Protocol* (BGP). Com isso, a demanda por processamento, armazenamento de dados e segurança das informações, aliada aos limites atuais da Internet e a crescente procura por serviços de computação em nuvem, fez surgir uma nova forma de organizar a infraestrutura para prover esses serviços, denominada *data center*.

Um *data center* é uma grande infraestrutura composta por milhares de elementos de redes (computadores, *switches*, *no-breaks*,...) que são organizados seguindo metodologias de interconexão, fornecimento de energia, refrigeração, etc. As redes de centro de dados em nuvem (*cloud data center*), estimuladas pelo crescente interesse na escalabilidade, custo e gerência, possuem requisitos específicos em relação ao funcionamento tradicional das interconexões inter-domínios ou redes *Local Area Network* (LAN)/*Wide Area Network* (WAN). Como consequência, os fornecedores de infraestrutura de *data center* estão muito interessados na otimização do projeto da rede, como escalar o acondicionamento dos equipamentos, uma tarefa árdua multidisciplinar que tem impulsionado propostas radicais para interligar computadores [1]. As propostas mais recentes incluem, por exemplo, a modularização de contêineres construídos sob medida, onde os servidores atuam como roteadores e os *switches* como equipamentos desprovidos de inteligência (por exemplo, BCube [2]), desenvolvimento de uma camada 2 virtual (por exemplo, VL2 [3]), ou localização baseada em pseudos endereços físico ou *Media Access Control* (MAC) (por exemplo, PortLand [4]).

Assim como as grades computacionais, a nuvem é um modelo de *utility computing* [5]. Esse tipo de computação envolve uma dinâmica no aumento e na redução dos recursos os quais, quando agregados, são apresentados aos clientes como um conjunto único de recursos de processamento e armazenamento, virtualmente "infinitos". Embora as aplicações em nuvem apresentadas para os usuários finais sejam interativas (por exemplo, buscas, E-mail, armazenamento de fotos e serviços de localização geográfica), essas aplicações e serviços não seriam possíveis sem as tarefas de análise, processamento e armazenamento de dados em grande escala (por exemplo, MapReduce [6], Hadoop [7], Dryad [8]). Um suporte adequado para esse conjunto de serviços heterogêneos na mesma infraestrutura impõe requisitos de rede tais como: (i) vazão e latência uniformes entre qualquer par de servidores para matrizes de tráfego altamente variáveis e dominadas por rajadas; (ii) suporte à mobilidade sem interrupção de cargas de trabalho; e (iii) a alocação ágil de máquinas virtuais em qualquer servidor físico disponível. Esses e outros requisitos da infraestrutura de Tecnologia da Informação (TI) impostos pelo modelo de *cloud computing* têm motivado as pesquisas de novos projetos e arquiteturas de redes adequadas para os *cloud data centers* de próxima geração [1].

Com o surgimento de *switches* com interface de programação de aplicações (*Application Programming Interface* (API)) aberta (*switches programáveis*) baseados na tecnologia OpenFlow [9],

emerge um promissor conceito de redes controladas via *software* ou, em inglês, *software-defined networking* [10]. O protocolo OpenFlow especifica um caminho padrão para controle das tomadas de decisões no tratamento dos pacotes (por exemplo, encaminhar para determinada porta, descartar, enviar para o controlador, etc.) através de uma inteligência (*software*) implementada em controladores logicamente centralizados (por exemplo, NOX [11] e Onix [12]), mantendo os fornecedores de equipamentos de rede responsáveis apenas pelo desenvolvimento do dispositivo (com suporte à tabela de fluxos especificada pelo OpenFlow), sem a necessidade de expor os detalhes de implementação do *hardware*. Nesse modelo de rede, a abordagem tradicional de gerência de baixo nível (por exemplo, endereços MAC e IP) e a operação de protocolos e algoritmos distribuídos (por exemplo, Bellman-Ford), tornam-se um problema "basicamente" de *software*, com visão global da rede (por exemplo, Dijkstra) e um maior nível de abstração (por exemplo, nomes de usuário, *Fully Qualified Domain Name* (FQDN), etc.). Dessa forma, a comunidade de desenvolvedores de sistemas distribuídos que tem contribuído com a realização dos denominados *mega data centers* ou *warehouse-scale computers* [13], pode definir o comportamento e novas funcionalidades da rede conforme os objetivos e requisitos específicos do provedor da infraestrutura e serviços, sem ter que depender dos demorados ciclos de atualização dos equipamentos de rede. Por isso, não é surpresa que provedores de infraestrutura para *cloud data centers*, tais como Amazon ou Google analisem com expressivo interesse a tecnologia OpenFlow. A grande razão para esse interesse reside nos níveis de controle e flexibilidade que as empresas necessitam na provisão dos serviços na nuvem a um custo inferior, mas com o mesmo desempenho dos *switches* comerciais atualmente em operação.

Motivados por esse cenário de desafios e oportunidades, foi projetada, implementada e validada uma proposta de arquitetura de rede de centro de dados (*Data Center Networks* (DCN)) que oferece um novo serviço de encaminhamento baseado na codificação de rotas na origem em um filtro de Bloom nos pacotes (*in-packet Bloom filter* (iBF)), no caso, adicionado aos campos de endereço do quadro MAC/Ethernet. A proposta apresenta um ambiente de implementação totalmente distribuído, com a inserção de dois serviços, escaláveis e tolerantes a falhas, para manter globalmente apenas as informações de topologia e diretório de servidores necessárias ao provimento do serviço de encaminhamento. Essas informações são mantidas em um sistema de armazenamento distribuído <chave,valor>. Com o desafio de fornecer uma arquitetura resiliente, o controle centralizado do paradigma OpenFlow/NOX utilizado na implementação da arquitetura proposta, denominada *Switching with in-packet Bloom filters* (SiBF) [14], torna-se um "exército" de *Rack Manager* (RM) (gerenciador de *rack*) com uma instância executando em cada *rack*. Essa abordagem oferece escalabilidade, desempenho e tolerância a falhas, aproveitando a disponibilidade e as características de bases de dados não relacionais tipicamente disponíveis em *data centers* (por exemplo, Amazon Dynamo [15]) como serviço de suporte ao desenvolvimento das aplicações em nuvem.

Os resultados deste trabalho sugerem que, suportado pela disponibilidade de *switches* programáveis de prateleira (*commodity*), um serviço típico de rede como, por exemplo, o encaminhamento de pacotes ou a própria base de dados utilizada no provimento desse encaminhamento, pode tornar-se mais um serviço da nuvem privada, oferecendo uma interligação eficiente dos servidores de aplicações e implementado segundo as boas práticas do desenvolvimento de aplicações distribuídas nos *cloud data centers* (baixa latência, confiabilidade e escalabilidade). Além disso, o trabalho questiona o argumento de que o modelo de controle centralizado, como nas redes OpenFlow, possui um único ponto de falha. E mostra ainda que os controladores de rede podem ser fisicamente distribuídos mas com uma visão centralizada dos recursos da rede, resultando em uma abordagem de rede intermediária, entre totalmente distribuída e centralizada.

1.1 Contribuições da Nova Arquitetura

1) Propõe um serviço de encaminhamento escalável e compatível com os *hardware* de rede existentes (equipamentos de prateleiras com OpenFlow-habilitado), não requer modificações na borda da rede, é auto-configurável e pode ser oferecido como um serviço alternativo de transmissão em paralelo com outras tecnologias Ethernet (protocolos, *middleboxes*, etc.);

2) Apresenta uma solução que mostra como o controle centralizado do paradigma OpenFlow/NOX pode ser implementado de forma distribuída, gerando um modelo intermediário entre o paradigma de rede totalmente distribuído e totalmente centralizado;

3) Argumenta que o encaminhamento de pacotes em si pode tornar-se um serviço interno do *data center* adequado às necessidades da aplicação e implementado seguindo as melhores práticas de programação em nuvem, como a adoção de sistemas de armazenamento de chave-valor.

4) Permite a separação efetiva entre localizador e identificador (espaço de nomes), através do mecanismo de encaminhamento utilizando rotas na origem codificadas em filtro de Bloom e inserido nos quadros (MAC/Ethernet), removendo a hierarquia e os problemas que o protocolo IP ocasiona nas redes de *data center*.

1.2 Contribuições do Autor

A arquitetura proposta e apresentada nesta dissertação (SiBF) foi concebida através de discussões e estudos do grupo de trabalho do Laboratório de Automação e Computação (LCA) da Unicamp, coordenado pelo professor Maurício F. Magalhães. Portanto, trata-se de um trabalho de pesquisa e desenvolvimento realizado em colaboração, o que exige um esclarecimento quanto a distribuição das contribuições. O projeto da arquitetura SiBF, isto é, os princípios e o método de encaminhamento

baseado em filtro de Bloom nos pacotes (iBF), tem origem no estudo de métodos de encaminhamento compactos baseados em estruturas de dados probabilísticas [16], sendo o cenário de *data center* um estudo de caso. As contribuições mais específicas do autor quanto à implementação da arquitetura estão listadas através das publicações a seguir.

1. Christian E. Rothenberg, Carlos A. B. Macapuna, Fábio L. Verdi, Maurício F. Magalhães, Alexander Zahemszky. "Data center networking with in-packet Bloom filters". *XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos* (SBRC 2010).

   Contribuições: O escopo desta dissertação começou a ser elaborado com a concepção desta publicação, onde vários princípios foram identificados no contexto de encaminhamento compacto de pacotes [16]. As principais contribuições do autor devem-se à especificação e implementação do procolo de descoberta de topologia assim como dos trabalhos de implementação de módulos do Gerenciador de Rack (RM). Além disso, o autor foi responsável pela criação do ambiente de experimentação (*testbed*) com suporte a *switches* OpenFlow capazes de tomar decisões de encaminhamento baseadas na presença de subconjuntos de bits nos cabeçalhos Ethernet (iBF), a auto-configuração do ambiente e a execução dos testes de alcançabilidade e balanceamento de carga.

2. Carlos A. B. Macapuna, Christian E. Rothenberg, Maurício F. Magalhães. "Controle distribuído em Redes de Data Center baseado em Gerenciadores de Rack e Serviços de Diretório/Topologia". *VIII Workshop em Clouds, Grids e Aplicações* (WCGA 2010).

   Contribuições: Após a implementação do serviço de encaminhamento utilizando filtro de Bloom nos pacotes e a concepção da arquitetura baseada na utilização do paradigma OpenFlow/NOX, surgiu um novo desafio: distribuir o controle centralizado do OpenFlow. Esse trabalho teve total participação do autor, onde o controle tornou-se logicamente centralizado, mas com uma gama de *Rack Managers* distribuídos fisicamente e operando de forma colaborativa com os serviços distribuídos de armazenamento de chave-valor da topologia da rede e do diretório de máquinas virtuais. Dessa forma, o autor contribuiu para tornar realidade o princípio de tolerância ampla à de falhas.

3. Carlos A. B. Macapuna, Christian E. Rothenberg, Maurício F. Magalhães. "In-packet Bloom filter based data center networking with distributed OpenFlow controllers". *IEEE International Workshop on Management of Emerging Networks and Services (IEEE MENS 2010).*

   Contribuições: Este trabalho apresenta para a comunidade internacional os principais pontos chaves da arquitetura proposta, como o controle distribuído utilizando o compartilhamento do estado global da rede, através de um Serviço de Topologia e um Serviço de Diretório implementados com bases de dados distribuídas do tipo chave-valor de fácil implementação e tornando

o sistema tolerante a falhas dos múltiplos componentes (*switches*, *Virtual Machine*s (VMs), controladores, nós da base de dados). Nessa publicação, o autor implementou mais um ambiente de testes, com o objetivo de obter um maior número de elementos nas simulações nos experimentos.

4. Christian E. Rothenberg, Carlos. A. B. Macapuna, Fábio L. Verdi and Maurício F. Magalhães. "The Deletable Bloom filter: A new member of the Bloom family". *IEEE Communications Letters*, vol.14, no.6, pp.557-559.

   Contribuições: O autor participou da análise teórica das avaliações da proposta de filtros de Bloom deletáveis. Essa proposta poderá ser adicionada na arquitetura SiBF em trabalhos futuros, onde utilizando a ideia de regiões, o identificador de *switch* (ou *middlebox*) é removido toda vez que um pacote for encaminhado utilizando o filtro de Bloom, uma maneira de reduzir os possíveis falsos positivos, minimizando o risco de *loops* e permitindo que novos elementos possam ser inseridos dinamicamente.

5. Christian E. Rothenberg, Carlos A. B. Macapuna, Maurício Ferreira Magalhães, Fábio L. Verdi, Alexander Wiesmaier, "In-packet Bloom filters: Design and networking applications", *Computer Networks*, In Press, Corrected Proof, Available online 19 Dec 2010.

   Contribuições: Neste trabalho o autor contribuiu na avaliação experimental de funções de *hash* que são utilizadas para adicionar elementos em um filtro de Bloom. Como principais resultados do estudo de diferentes famílias de funções de *hash*, assim como a multiplicidade de parâmetros dos filtros de Bloom, destacam-se a validação experimental de técnicas de fácil implementação e o baixo consumo de recursos.

1.3 Organização da dissertação

O restante desta dissertação está estruturado em mais 4 capítulos, conforme descrito a seguir. O Capítulo 2 apresenta as principais tecnologias e propostas relacionadas com a dissertação. O Capítulo 3 detalhada as informações relacionadas à nova arquitetura proposta, assim como os princípios adotados. O Capítulo 4 descreve os principais pontos correspondentes à implementação do protótipo, além de apresentar a validação e a avaliação do protótipo em relação a tolerância a falhas, balanceamento de carga e desempenho face a possíveis falsos positivos. Finalmente, o Capítulo 5 apresenta as conclusões e discute os trabalhos futuros.

Capítulo 2

Tecnologias e Trabalhos Relacionados

Este capítulo discute as mais importantes tecnologias envolvidas no desenvolvimento e implementação do trabalho apresentado nesta dissertação. Além disso, faz uma revisão bibliográfica de arquiteturas e trabalhos relacionados. O capítulo encontra-se divido em duas seções: a primeira apresenta as tecnologias que dão suporte à programabilidade em redes de computadores, através de APIs abertas como a tecnologia OpenFlow juntamente com o controlador NOX, que agregados, tornam-se fundamentais na concepção da arquitetura SiBF; a segunda descreve os pontos essenciais de outras propostas recentes relacionadas com os métodos e princípios adotados.

2.1 Programabilidade na Rede

Um ambiente real para experimentação de novas propostas têm sido foco de grandes projetos de redes (GENI [17], PlanetLab [18]). Tal importância se deve à crescente necessidade em obter resultados mais próximos do ambiente de produção. As novas tecnologias OpenFlow/NOX e Open vSwitch (Figura 2.1) seguem essa tendência, disponibilizando um substrato de programabilidade através de equipamentos de redes com APIs abertas, que podem ser utilizados em redes de pesquisa e produção, sem a necessidade do fabricante expor os segredos de implementação dos seus produtos. Os equipamentos de redes programáveis proporcionam liberdade para o pesquisador desenvolver seus experimentos e colocar em prática suas ideias, acelerando a criação de novos protótipos de *software* e *hardware* de rede. Além disso, com a crescente demanda e o baixo custo dos equipamentos de prateleiras (*comoditização*), a infraestrutura de rede está sendo redesenhada e novos módulos estão sendo inseridos como, por exemplo, novos elementos de decisão. Nesta seção são descritos os principais conceitos para elaboração de ambientes com equipamentos programáveis e que podem ser implementados nas redes utilizadas no dia-a-dia.

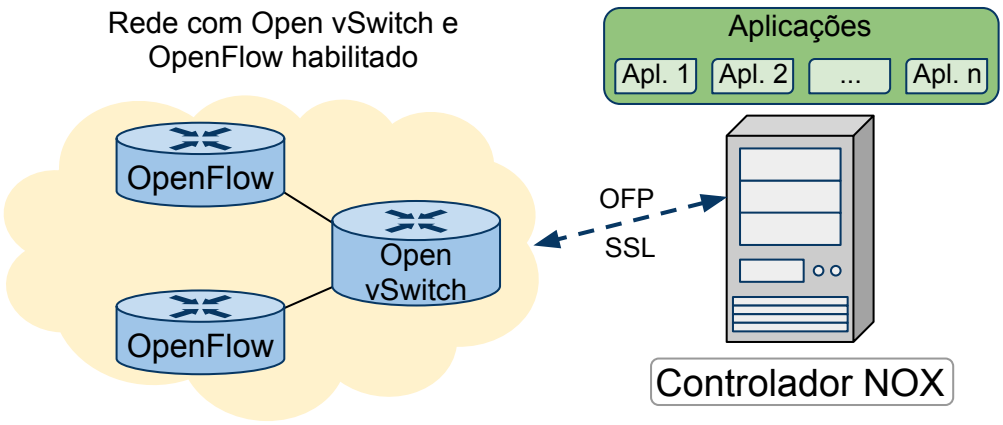


Fig. 2.1: Elementos de uma rede baseada no OpenFlow, Open vSwitch e NOX.

2.1.1 OpenFlow

O OpenFlow é uma proposta baseada no modelo de controle de rede logicamente centralizado, que permite ao pesquisador executar seus experimentos em redes (virtuais, de pesquisas ou utilizadas no dia-a-dia) sem interferir no tráfego de produção. Baseado em *switches Ethernet* comerciais, OpenFlow define um protocolo padrão para que elementos controladores possam acessar remotamente e modificar o estado dos equipamentos (*switches*, roteadores, pontos de acesso, etc.).

A contribuição mais importante do paradigma do OpenFlow é a generalização do plano de dados. Ou seja, qualquer modelo de encaminhamento de dados baseado na tomada de decisão fundamentada em algum valor do campo de cabeçalho dos pacotes (nas camadas 2, 3 ou 4) pode ser suportado. As entradas correspondentes na tabela de fluxos podem ser interpretadas como decisões em *cache* (*hardware*) do plano de controle (*software*).

Um fluxo na rede é portanto nada mais que a mínima unidade controlável para criar sistemas escaláveis usando OpenFlow. Trata-se de um compromisso pragmático que permite pesquisadores realizarem experimentos em suas redes sem exigir que os fabricantes revelem o funcionamento interno dos equipamentos. Para isso, o OpenFlow explora as funções (campos) comuns das tabelas de fluxos dos *switches* atuais que são utilizadas para implementar serviços como *Network Address Translation* (NAT), *firewall* e *Virtual Local Area Network*s (VLANs), o objetivo é oferecer suporte ao maior número de equipamentos. De acordo com a especificação do OpenFlow (versão 1.0), os campos suportados são listados na Figura 2.1. A administração de redes pode ser apresentada como um bom exemplo de utilização na combinação dos campos (*matching*) para instalar regras em uma rede OpenFlow. Nesse caso, serviços de aplicações podem ser monitorados utilizando os campos `src port` e `dst port` do *Transmission Control Protocol* (TCP)/*User Datagram Protocol* (UDP) para oferecer, por exemplo, qualidade de serviços ou barrar determinadas aplicações.

Tab. 2.1: Campos de pacotes utilizados pelo OpenFlow.

Ingress Port	Ether src	Ether dst	Ether type	VLAN id	VLAN priority	IP src	IP dst	IP proto	IP ToS bits	TCP/UDP src port	TCP/UDP dst port

Basicamente, o paradigma OpenFlow é composto por três partes (cf. Figura 2.2):

1. **Tabela de Fluxos**: A tabela de fluxos contém um conjunto de entradas com uma ação associada. Cada entrada na tabela consiste em **campos do cabeçalho** (utilizados para definir um fluxo), **ações** (definem como os pacotes devem ser processados e para onde devem ser encaminhados) e **contadores** (utilizados para estatísticas ou remoção de fluxos inativos).

2. **Canal Seguro**: Para que a rede não sofra ataques de elementos mal intencionados, o *Secure Channel* garante confiabilidade na troca de informações entre o *switch* e o controlador. A interface de acesso ao tráfego utilizada é o protocolo *Secure Socket Layer* (SSL). O OpenFlow também suporta outras interfaces (passivas ou ativas) para estabelecer a comunicação sem a necessidade de criptografia como, por exemplo, conexões TCP e *Packet capture* (Pcap). Essas interfaces são úteis em ambientes de testes devido à simplicidade de utilização já que não necessitam de chaves criptográficas.

3. **Protocolo OpenFlow**: O *switch* com Openflow habilitado disponibiliza um protocolo aberto para estabelecer comunicação externa com o controlador. Dessa forma, ao fornecer uma interface que atue sobre os fluxos de um *switch*, o Protocolo OpenFlow evita a necessidade de equipamento de rede programável, como NetFPGA. Basicamente, o controlador atua sobre as tabelas de fluxos dos *switches* através do canal seguro e, o protocolo OpenFlow é usando para realizar as tomadas de decisões de acordo com as regras definidas para cada entrada na tabela de fluxos (ações).

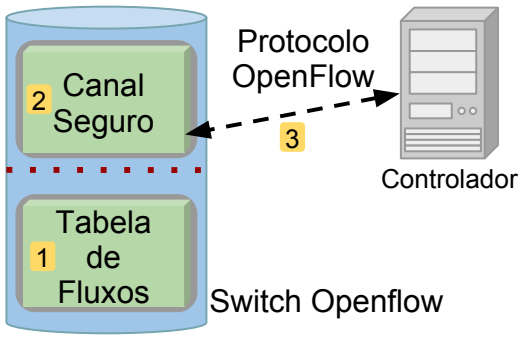


Fig. 2.2: As três partes que compõem o OpenFlow: tabela de fluxo, canal seguro e protocolo.

Com isso, um *switch* OpenFlow é um elemento de rede que encaminha pacotes entre portas, definido por um processo remoto, ou seja, a tomada de decisão depende das regras definidas pelas aplicações do controlador. Sendo assim, os fluxos são limitados pela capacidade de determinada aplicação. Um fluxo pode ser, por exemplo, todos os pacotes em uma faixa de endereços IPs, ou uma conexão TCP em uma porta específica, ou ainda todos os pacotes pertencentes a uma mesma VLAN. Para cada novo fluxo de pacotes que chega ao *switch*, uma ação é associada, que pode ser (1) encaminhar para uma determinada porta, (2) encapsular e transmitir para o controlador, (3) descartar, (4) reescrever os cabeçalhos ou (5) enviar para o processamento normal do *switch* nas camadas 2 ou 3. Essas e outras questões são definidas por um consócio, denominado *The OpenFlow Switch Consortium* [9], que está aberto à comunidade para discussões sobre a especificação do protocolo.

2.1.2 NOX

O gerenciamento de redes é um dos vários campos onde a atual Internet necessita de soluções mais eficazes, isto porque ele é realizado em baixo nível, distribuído nos elementos de rede (protocolos). Além disso, as configurações normalmente são feitas manualmente, ocasionando erros e aumentando a complexidade da rede. Assim como os Sistemas Operacionais proporcionaram um avanço no gerenciamento das aplicações, surge a necessidade de um Sistema Operacional de Redes com a capacidade de analisar e gerenciar a rede como um todo [11]. Nesse contexto, foi proposto o NOX [11] como um sistema para gerenciar e controlar as redes de campus caracterizado pela reutilização de componentes e pela especificação de uma API de controle. Esse paradigma permite a evolução em paralelo das tecnologias nos planos de dados e as inovações na lógica das aplicações.

O NOX está disponível gratuitamente traduzindo-se em um importante *framework* para construir novas aplicações para interação com os dispositivos que possuem o OpenFlow habilitado. A interface de programação do NOX é construída sobre os eventos, seja por componentes principais do NOX (*core*), ou definidos por usuários e gerenciados diretamente a partir de mensagens OpenFlow do tipo `packet-in`, `switch join`, `switch leave`. Um **componente** é um encapsulamento de funcionalidades que são carregadas com o NOX, que podem ser escritos em Python ou C++. Um **evento** é uma ação realizada sobre um determinado fluxo, onde as aplicações NOX utilizam um conjunto de *handlers* (manipuladores) que são registrados para serem executados quando da ocorrência de um evento específico. O OpenFlow limita-se à definição do protocolo de comunicação entre o *switch* e o controlador, cabendo ao controlador a responsabilidade de adicionar e remover as entradas na tabela de fluxos de acordo com o objetivo desejado da aplicação (componente). O NOX serve como uma camada de abstração para criar as aplicações e serviços que irão gerenciar as entradas de fluxos nos *switches* OpenFlow.

2.1.3 Open vSwitch

O *Open vSwitch* [19] é um *switch* de rede desenvolvido especificamente para ambientes virtuais. Trata-se de um *software* que fornece conectividade entre máquinas virtuais e interfaces físicas (ou virtuais). O objetivo do projeto é construir um *switch* para máquinas virtuais com suporte às interfaces padrões de gerenciamento. Um ponto forte dessa tecnologia é que, por ser escrita inteiramente em *software*, não está atrelada à rigidez imposta pelo *hardware* dos *switches*, sendo muito mais flexível e de rápido desenvolvimento. Dentre os principais casos de uso podemos citar o gerenciamento centralizado, a mobilidade entre sub-redes e a implementação de *Virtual Private Network*s (VPNs). Uma questão importante é que o Open vSwitch oferece suporte nativo ao protocolo OpenFlow.

2.2 Propostas de Arquitetura de Data center

As aplicações contidas nos *data centers* atuais sofrem com a fragmentação interna dos recursos e com a rigidez e limitação de banda que são impostas pela arquitetura de rede que conecta os servidores. Por exemplo, arquiteturas convencionais de rede utilizam mecanismos estáticos para mapear os serviços oferecidos pelo *data center* em VLANs, limitadas em algumas centenas de servidores devido ao nível de sinalização (*overhead*) gerado no plano de controle da rede (*Address Resolution Protocol* (ARP), *flooding*, *Dynamic Host Configuration Protocol* (DHCP)). Além disso, a utilização de equipamentos no nível IP para efetuar a distribuição de tráfego entre diversas VLANs, em conjunto com os balanceadores de carga necessários para espalhar as requisições entre os servidores, elevam o custo de implantação dos *data centers*. Basicamente, esse cenário é caracterizado pela concentração de tráfego em alguns poucos equipamentos, resultando em frequentes substituições de hardware para atender a novas demandas do *data center*. Pesquisas recentes têm apresentado propostas para contornar esses e outros problemas, como veremos a seguir.

2.2.1 Monsoon

A filosofia defendida pelo Monsoon [20] é a *comoditização* da infraestrutura como forma de obter escalabilidade e baixo custo, ou seja, o Monsoon adiciona equipamentos novos e baratos (*scale-out*) para atender a nova demanda, ao invés da constante troca por equipamentos de maior capacidade de processamento (*scale up*). O Monsoon estabelece uma arquitetura de *data center* organizada em formato de malha com o objetivo de comportar 100.000 ou mais servidores. Para criar essa malha são utilizados *switches* programáveis de camada 2 (*comoditizados*) e servidores. Modificações no plano de controle dos equipamentos são necessárias para suportar, por exemplo, roteamento com rota na origem. As modificações no plano de dados, por sua vez, visam suportar o encaminhamento de

dados por múltiplos caminhos através do *Valiant Load Balancing* (VLB). Nessa proposta, a função de balanceamento de carga é compartilhada por um grupo de servidores. Consequentemente, os equipamentos responsáveis pelo balanceamento de carga podem ser distribuídos pelos *racks* do *data center*, oferecendo maior agilidade e menor fragmentação na utilização dos recursos disponíveis.

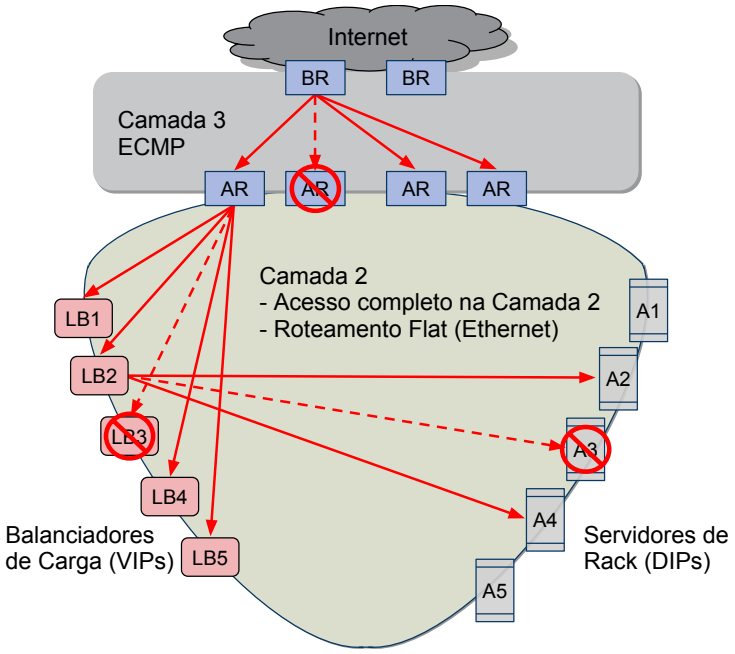


Fig. 2.3: Visão geral da arquitetura do Monsoon. Redesenhada de [20].

A Figura 2.3 apresenta uma visão geral da arquitetura do Monsoon. Os dois principais aspectos da arquitetura são: (1) a definição de uma única rede de camada 2 na qual todos os 100.000 servidores são conectados e (2) a flexibilidade pela qual as requisições podem ser distribuídas entre os diversos conjuntos de servidores. Se observadas a partir de uma perspectiva arquitetural de mais alto nível, as diferenças entre as camadas 2 (Ethernet) e 3 (IP) estão diminuindo, especialmente para redes contidas dentro de um mesmo prédio. Entretanto, existem alguns fatores práticos que levaram o Monsoon a optar pela camada 2 como tecnologia para criar um único domínio no qual todos os servidores estão conectados: (1) cortar custos; (2) eliminar a fragmentação de servidores e (3) diminuir o distúrbio às aplicações. Considerando esses fatores, o Ethernet é claramente a tecnologia eleita, pois está abaixo do IP e já apresenta custo e desempenho otimizados para o encaminhamento baseado em endereços planos.

Como observado na Figura 2.3, o domínio de camada 2 provê total conectividade entre os servidores do *data center*. A comunicação entre os servidores utiliza a taxa máxima das interfaces de rede, sem que ocorram enlaces sobrecarregados. A porção da rede que utiliza a camada 3 é necessária para conectar o *data center* à Internet, utilizando roteadores de borda (*Border Router*s (BRs)) e o

Equal Cost MultiPath (ECMP) para espalhar as requisições igualmente entre os roteadores de acesso (*Access Router*s (ARs)). Assim que as requisições alcançam o *data center*, os roteadores de acesso utilizam técnicas de *hash* consistente para distribuir uniformemente as requisições entre os diversos balanceadores de carga (*Load Balancer*s (LBs)) que estão associados a um endereço IP virtual (*Virtual IP* (VIP)) de uma determinada aplicação visível publicamente. Finalmente, os balanceadores de carga utilizam funções de distribuição específicas de cada aplicação para espalhar as requisições dentre o conjunto de servidores identificados pelo endereço IP (*Direct IP* (DIP)) associado ao serviço requisitado.

A Figura 2.3 também indica a existência de um mecanismo de recuperação de falhas, sejam elas em qualquer dos roteadores de acesso, balanceadores de carga ou servidores. Um serviço de recuperação (*health service*) continuamente monitora o estado de todos os equipamentos que constituem o *data center*. Por exemplo, um servidor que venha a falhar é imediatamente removido do conjunto de servidores associados a uma determinada aplicação, evitando que novas requisições sejam encaminhadas para o servidor em falha.

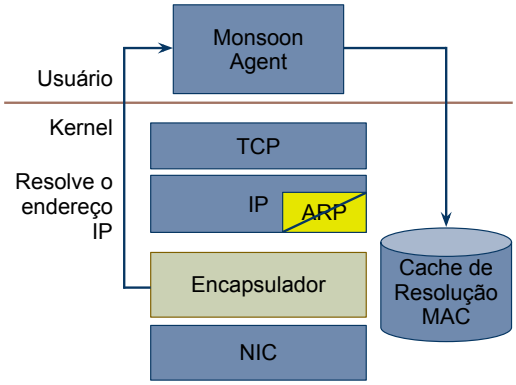


Fig. 2.4: Pilha de rede implementada pelos servidores do Monsoon. Redesenhada de [20].

O Monsoon utiliza tunelamento MAC-in-MAC para encaminhar pacotes entre os servidores que constituem o *data center*. Para tanto, é utilizado um serviço de diretório no qual a lista de endereços MAC dos servidores responsáveis pelas requisições, bem como o endereço MAC dos *switches* nos quais os servidores estão conectados, é mantida. A Figura 2.4 apresenta a pilha de rede implementada pelos servidores do Monsoon. Nessa pilha, a tradicional função de ARP é desativada e substituída por um processo executado em espaço de usuário (*Monsoon Agent*) e uma nova interface MAC virtual (*Encapsulador*). Note que essas mudanças são imperceptíveis para as aplicações. Basicamente, o encapsulador recebe pacotes vindos da camada de rede (IP) e consulta seu cache de resoluções MAC em busca da informação de encaminhamento. Caso não exista, o encapsulador solicita ao agente uma nova resolução através do **serviço de diretório**. O serviço de diretório resolve o endereço IP de destino e retorna uma lista contendo todos os endereços MAC dos servidores associados ao serviço

solicitado. Com base nessa lista, o encapsulador escolhe um servidor (seleciona seu MAC), encontra o MAC do *switch* de topo de *rack* (*Top of Rack* (ToR)) no qual o servidor está conectado e, finalmente, escolhe um *switch* intermediário no qual os pacotes serão enviados (*bounce off*) por questões de balanceamento de carga do VLB. Esse conjunto de informações corresponde a um único fluxo de dados e é mantido em cache para que todos os quadros do fluxo recebam o mesmo tratamento.

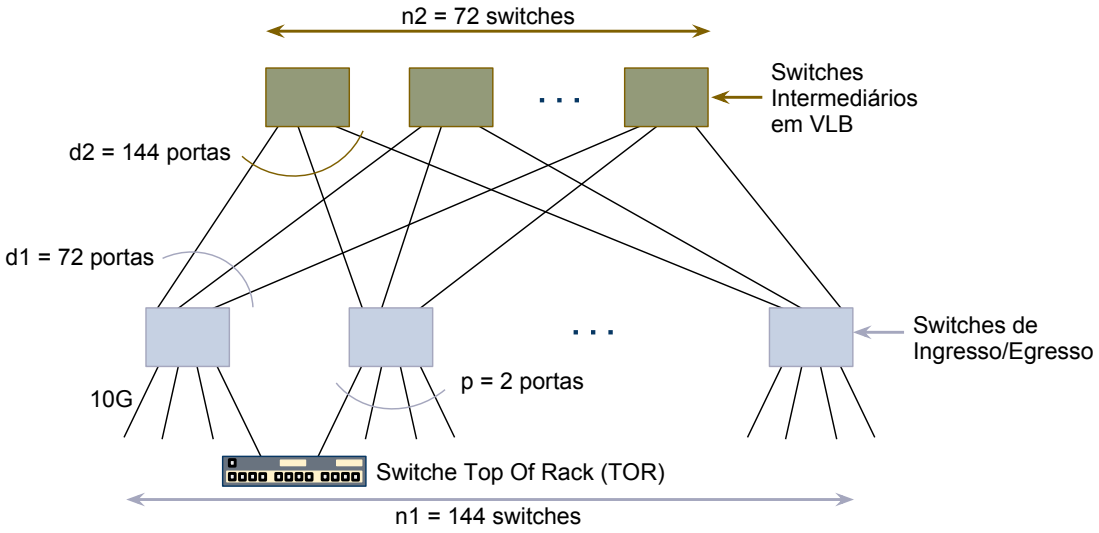


Fig. 2.5: Exemplo de topologia conectando 103.680 servidores. Redesenhada de [20].

A Figura 2.5 apresenta um exemplo concreto de uma topologia utilizada pelo Monsoon capaz de conectar 103.680 servidores em um único domínio de camada 2. Nesse cenário, cada *switch* ToR possui 2 portas Ethernet de 10Gbps que são conectadas a dois *switches* diferentes de ingresso/egresso localizados na região central (*core*) do *data center*, por questões de tolerância a falhas. A região central é organizada em dois níveis (*n1* e *n2*) de *switches*. Nesse exemplo, o nível *n1* possui 144 *switches* que não possuem qualquer ligação entre eles, mas cada um deles está conectado a todos os *switches* intermediários (nível *n2*) através de portas Ethernet de 10 Gbps. Sendo assim, existem 72 *switches* no nível *n2* (intermediários), tornando a topologia interessante para o VLB, pois os fluxos podem escolher os *switches* intermediários nos quais eles irão alcançar a partir desse conjunto com 72 *switches*. Finalmente, considerando-se o número de portas e *switches* presentes no nível *n1*, essa topologia é capaz de conectar 5184 *racks* com 20 servidores cada, totalizando 103.680 servidores em um único domínio de camada 2, possibilitando que cada servidor transmita à taxa máxima de sua interface (1 Gbps).

2.2.2 VL2

O *Virtual Layer 2* (VL2) é uma proposta de nova arquitetura de *data center* e pertence ao mesmo grupo de pesquisa que originou o Monsoon. Sendo assim, o VL2 pode ser considerado como uma evolução do Monsoon, introduzindo alguns refinamentos. O VL2 [3] trata as limitações citadas no início da seção (2.2) através da criação de uma camada 2 virtual. Essa camada virtual provê aos serviços do *data center* a ilusão de que todos os servidores associados a eles, e apenas os servidores associados a eles, estão conectados através de um único *switch* de camada 2 (livre de interferências) e mantém essa ilusão mesmo que o tamanho de cada serviço varie entre 1 ou 100.000 servidores. Para atingir essa meta, é preciso construir uma rede que honre três objetivos: (1) a comunicação servidor-a-servidor só pode ser limitada pela taxa de transmissão das interfaces de rede de cada servidor (1 Gbps); (2) o tráfego gerado por um serviço deve ser isolado de tal forma a não afetar o tráfego de outros serviços e (3) o *data center* deve ser capaz de alocar qualquer servidor para atender a qualquer serviço (agilidade), possibilitando a atribuição de qualquer endereço IP àquele servidor, de acordo com as exigências do serviço e independente da sua localização no *data center*.

Além de propor uma arquitetura que busca prover agilidade na alocação de servidores, o VL2 investiga que tipo de solução poderia ser construída utilizando os recursos disponíveis atualmente, evitando qualquer tipo de modificação no hardware de *switches* e servidores e oferecendo, ainda, um cenário transparente para aplicações legadas. Sendo assim, uma prática comum em *data centers* é a modificação do *software* (sistemas operacionais) utilizados nos servidores. Nesse contexto, o VL2 propõe uma reorganização nos papéis desempenhados tanto pelos servidores quanto pela rede, através da introdução de uma camada de *software* (*shim*) na pilha de protocolos implementada pelos servidores, de tal forma a contornar as limitações impostas pelos dispositivos legados de rede.

A Figura 2.6 apresenta a topologia utilizada pelo VL2, um *backbone* com elevada conectividade entre os *switches* de agregação e intermediários. Os *switches* ToR são conectados a dois *switches* de agregação e, devido ao elevado número de conexões disponíveis entre qualquer par de *switches* de agregação, a falha de qualquer um dos s [1] *switches* intermediários reduz em apenas $1/s$ a largura de banda disponível, garantindo uma lenta degradação do serviço oferecido pelo *data center*.

A rede é constituída por duas classes de endereços, os endereços com significado topológico (*Locator Addresses* (LAs)) e os endereços planos de aplicação (*Application Addresses* (AAs)). Nesse cenário, a infraestrutura de rede utiliza endereços LA, que são atribuídos para todos os *switches* e suas interfaces. Além disso, todos os *switches* executam um protocolo de roteamento baseado no estado do enlace para disseminar esses LAs, oferecendo uma visão global da topologia formada pelos *switches* e possibilitando o encaminhamento de pacotes encapsulados em LAs através de caminhos mais curtos. Por outro lado, as aplicações utilizam AAs que permanecem inalterados, independente-

[1] Número total de *switches* intermediários.

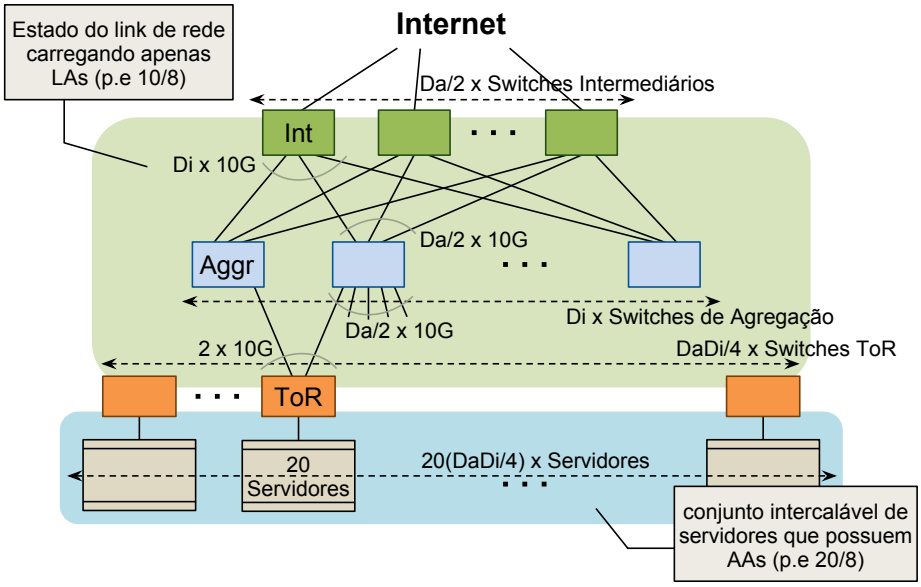


Fig. 2.6: Exemplo de *backbone* utilizado pelo VL2. Redesenhada de [3].

mente da maneira como os servidores migram no interior do *data center*. Para todo AA é associado o LA atribuído ao *switch* ToR no qual o servidor está conectado. Esse mapeamento é mantido por um serviço de diretório do VL2.

A malha de camada 3 formada pelo VL2 cria a ilusão de um único domínio de camada 2 para os servidores no interior do *data center*, uma vez que os servidores imaginam pertencer a uma mesma VLAN. Note que todos os servidores na Figura 2.6 possuem um endereço AA alocado a partir da faixa 20/8. Nesse cenário de camada 3, as requisições vindas da Internet podem fluir diretamente até os servidores, sem serem forçadas através de *gateways* específicos nos quais os pacotes são reescritos. Para tanto, endereços LA adicionais são atribuídos aos servidores a partir de uma faixa de IPs válidos (alcançáveis) na Internet.

Basicamente, para rotear tráfego entre servidores identificados por endereços AA em uma rede que possui rotas formadas a partir de endereços LA, um agente VL2 executando em cada um dos servidores intercepta os pacotes originados e os encapsula em pacotes endereçados ao LA do *switch* ToR associado ao servidor de destino. Existe ainda um terceiro cabeçalho encapsulando todos os pacotes por razões de espalhamento de tráfego. O sucesso do VL2 está associado ao fato dos servidores acreditarem compartilhar uma única sub-rede IP, devido ao encapsulamento efetuado. Como forma de contornar o *broadcast* gerado pelo protocolo ARP, na primeira vez em que um servidor envia pacotes para um determinado endereço AA, a pilha de rede original do servidor gera uma requisição ARP e a envia para a rede. Nesse instante, o agente VL2 presente no servidor intercepta a requisição ARP e a converte em uma pergunta *unicast* para o serviço de diretório do VL2. O serviço de diretório, por sua vez, responde com o endereço LA do ToR de destino no qual os pacotes devem ser tunelados e

o agente VL2 armazena esse mapeamento através de um procedimento similar ao cache original do ARP evitando, dessa forma, novas perguntas ao serviço de diretório.

O VL2 combina o VLB e o ECMP para evitar áreas de concentração de tráfego. O VLB distribui o tráfego entre um conjunto de *switches* intermediários e o ECMP é utilizado para espalhar o tráfego entre caminhos de custo igual. A combinação de ambos os mecanismos cria um cenário de distribuição de tráfego mais eficaz, uma vez que as limitações presentes em um mecanismo são tratadas pelo outro. Em suma, entre a origem e o destino, o pacote é enviado aos *switches* intermediários, desencapsulado por esses *switches*, encaminhado para o ToR de destino, desencapsulado novamente e, finalmente, enviado ao destinatário. Esse processo de encapsulamento de pacotes na direção de um *switch* intermediário satisfaz o VLB. Entretanto, eventuais falhas nesses *switches* intermediários poderiam levar a um cenário no qual um elevado número de agentes VL2 teriam de ser atualizados para convergir ao novo estado da rede. O VL2 contorna essa situação atribuindo o mesmo endereço LA para todos os *switches* intermediários. Note que na topologia adotada pelo VL2, todos os *switches* intermediários estão a exatos três saltos de distância dos servidores de origem, criando o cenário adequado para a utilização do ECMP. Sendo assim, o ECMP assume a responsabilidade de entregar os pacotes para um dos *switches* intermediários e, em caso de falhas, o ECMP reage, enviando os pacotes para um *switch* que esteja operacional, eliminado a necessidade de avisar os diversos agentes VL2.

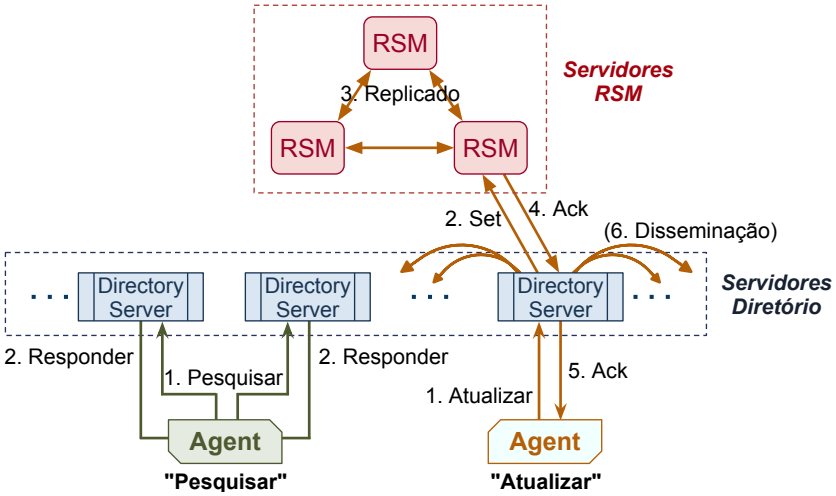


Fig. 2.7: Arquitetura do serviço de diretório do VL2. Redesenhada de [3].

O serviço de diretório do VL2 provê três serviços principais: (1) consultas; (2) atualizações de mapeamentos entre AAs e LAs e (3) um mecanismo de atualização de cache reativo para atualizações sensíveis a atrasos (por exemplo, a atualização entre um AA e um LA durante o processo de migração de uma máquina virtual). Considerando os requisitos de desempenho e os padrões de consultas e

atualizações, o VL2 define uma arquitetura de dois níveis conforme ilustra a Figura 2.7. O primeiro nível, considerando-se um *data center* com aproximadamente 100.000 servidores, possui entre 50 e 100 servidores de diretório (*Directory Servers*) otimizados para leitura (consultas), utilizados para replicar os dados do serviço de diretório e responder a consultas dos agentes VL2. No segundo nível, existe um número pequeno, entre 5 e 10, de servidores otimizados para escrita (*Replicated State Machine* (RSM)), que oferecem um serviço consistente de armazenamento. Cada servidor de diretório possui em *cache* todos os mapeamentos AA-LA disponíveis nos servidores RSM e responde às consultas feitas pelos agentes VL2 de forma independente. Para esses servidores de diretório, não há a necessidade do oferecimento de elevados níveis de consistência. Sendo assim, as sincronizações com os servidores RSM ocorrem a cada 30 segundos. Em caso de atualizações na rede, o sistema envia a nova informação para um *Directory Server* que, por sua vez, encaminha a atualização para um servidor RSM. O servidor RSM replica essa informação entre todos os servidores RSM e, finalmente, envia uma mensagem de confirmação para o servidor de diretório que originou a atualização.

2.2.3 PortLand

O PortLand [4] é um conjunto de protocolos compatíveis com Ethernet para efetuar roteamento, encaminhamento e resolução de endereços. É desenvolvido considerando-se a estrutura organizacional comumente encontrada em *data centers*, ou seja, uma árvore com múltiplos nós na raiz (denominada *fat tree*). O PortLand propõe: (1) a utilização de um protocolo para possibilitar que os *switches* descubram sua posição (topológica) na rede; (2) a atribuição de Pseudo endereços MAC (*Pseudo MAC* (PMAC)) para todos os nós finais, de forma a codificar suas posições na topologia; (3) a existência de um serviço centralizado de gerenciamento da infraestrutura de rede (*Fabric Manager*) e (4) a implantação de um serviço de *Proxy* para contornar o *broadcast* inerente ao ARP.

A Figura 2.8 ilustra uma topologia *fat tree* em três níveis utilizada pelo PortLand. Para construir uma topologia em três níveis como essa, é preciso estabelecer o parâmetro p_n que define o número de portas em cada *switch* ($p_n = 4$ neste exemplo). Em geral, a topologia em três níveis constituída de *switches* com p_n portas suporta comunicação não-bloqueante entre ${p_n}^3/4$ servidores utilizando $5{p_n}^2/4$ *switches* de p_n portas. A topologia como um todo é organizada em p_n conjuntos de servidores (chamados de *pods*), nos quais é possível prover comunicação não-bloqueante entre ${p_n}^2/4$ servidores através de técnicas de *hash* e distribuição do ECMP. Do ponto de vista organizacional, a rede *fat tree* é relativamente fixa, possibilitando a construção e manutenção de *data centers* modulares, nos quais a filosofia de expansão é adicionar mais equipamentos (*racks* ou colunas de servidores) sob demanda. Obviamente, toda expansão requer uma fase anterior de planejamento na qual a estrutura de *switches* é definida de forma a suportar tais evoluções.

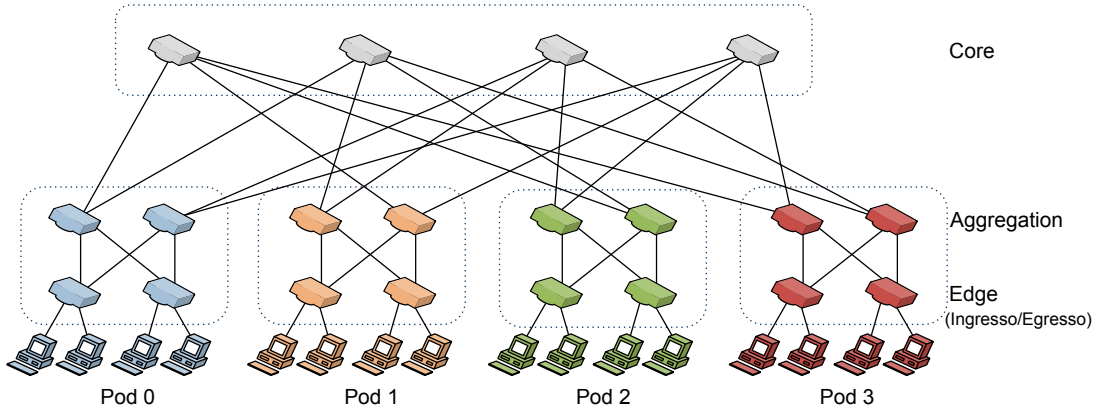


Fig. 2.8: Exemplo de topologia (*fat tree*) utilizada pelo PortLand. Redesenhada de [20].

O PortLand utiliza um gerenciador de infraestrutura de rede centralizado (denominado *Fabric Manager*) para manter estados relacionados à configuração da rede, tal como a sua topologia. O *Fabric Manager* é um processo executado no espaço do usuário em uma máquina dedicada responsável pelo tratamento das requisições ARP, tolerância a falhas e operações de *multicast*. De acordo com a especificação do PortLand, o *Fabric Manager* pode ser desenvolvido como um servidor conectado de forma redundante à estrutura do *data center* ou, ainda, ser executado em uma rede de controle separada.

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

No instante em que um *switch* de ingresso (*Edge*) observa a existência de um novo endereço MAC, os pacotes com esse endereço são desviados para o plano de controle do *switch*, que cria uma nova entrada na tabela de mapeamento e, na sequência, encaminha esse novo mapeamento para o *Fabric Manager* para futuras resoluções, conforme ilustra a Figura 2.9 (Passos 1a, 2a e 2b). Basicamente, o PortLand efetua a separação entre localizador/identificador de forma totalmente transparente aos nós finais e compatível com o hardware dos *switches comoditizados* disponíveis no mercado. Outra característica importante do PortLand refere-se à não utilização de técnicas de tunelamento para en-

caminhar os pacotes, sendo apenas necessário a reescrita de endereços PMAC/AMAC nas bordas do *data center* (*switches Edge*).

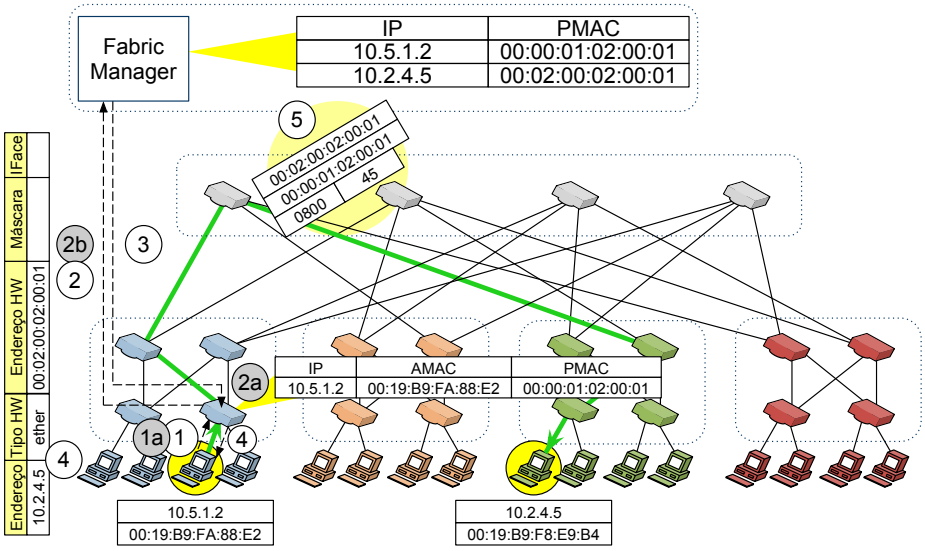


Fig. 2.9: Resoluções ARP utilizando mecanismo de *Proxy*. Adaptada de [20].

As requisições ARP, originalmente, efetuam *broadcast* e atingem todos os nós localizados em um mesmo domínio de camada 2. O PortLand utiliza o *Fabric Manager* para contornar o *overhead* de sinalização causado pelo ARP conforme ilustra a Figura 2.9. No passo 1, o *switch* de ingresso detecta a chegada de uma mensagem ARP requisitando um mapeamento IP para MAC, intercepta essa mensagem e a encaminha para o *Fabric Manager* no passo 2. O *Fabric Manager* consulta sua tabela de PMACs em busca do mapeamento e retorna o PMAC para o *switch* requisitante no passo 3. O *switch* de borda, por sua vez, cria uma mensagem de resposta do ARP e a retorna para o nó que originou a requisição no passo 4. O passo 5 mostra um quadro que representa a comunicação entre dois nós utilizando os endereços PMACs, origem e destino.

Existe um detalhe adicional para prover suporte à migração de máquinas virtuais. Assim que a migração é completada, ou seja, a máquina virtual termina sua transição entre um servidor físico e outro, a máquina virtual envia um ARP gratuito contendo seu novo mapeamento entre endereços IP e MAC. Esse ARP gratuito é encaminhado até o *Fabric Manager* pelo *switch* de borda. Infelizmente, os nós que estavam se comunicando com essa máquina virtual antes da migração manterão o mapeamento antigo em sua memória cache e terão de esperar até que o mapeamento expire para prosseguir com a comunicação. Entretanto, o *Fabric Manager* pode encaminhar uma mensagem de invalidação de mapeamento ao *switch* no qual a máquina virtual estava associada. Dessa maneira, o *switch* seria capaz de replicar o ARP gratuito aos nós que continuam originando pacotes na direção da máquina virtual que migrou, atualizando seus mapeamentos.

Os *switches* utilizam informações relativas às suas posições na topologia global do *data center* para efetuar encaminhamento e roteamento mais eficientes através da comunicação em pares, ou seja, encaminhamento considerando apenas os vizinhos diretamente conectados. Com o objetivo de criar um cenário *plug-and-play*, o PortLand propõe a utilização de um protocolo para descobrir a localização dos *switches* de forma automática. Nesse protocolo, chamado *Location Discovery Protocol* (LDP), os *switches* enviam, periodicamente, *Location Discovery Messages* (LDMs) em todas as suas portas para: (1) definir suas posições e (2) monitorar o estado de suas conexões físicas. Em resumo, o LDP consegue definir quais são os *switches* de borda (*Edge*), uma vez que eles recebem LDMs em apenas uma fração de suas portas, isto é, naquelas conectadas aos *switches* de agregação (*Aggregation*), pois os nós finais não geram LDMs.

A partir do momento que os *switches* de borda descobrem sua localização (nível) na topologia, as LDMs disseminadas subsequentemente passam a conter informação referente ao seu nível. Dessa forma, o restante dos *switches* são capazes de aferir suas respectivas posições. Os *switches* de agregação aferem sua posição, uma vez que eles recebem LDMs em todas as suas portas, sendo algumas originadas pelos *switches* de borda (contendo informação de nível) e as restantes originadas pelos *switches* de núcleo (sem informação de nível). Finalmente, os *switches* de núcleo (*core*) inferem as respectivas posições uma vez que todas as suas portas passarão a receber LDMs originadas por *switches* de agregação (contendo informação de nível). Uma vez definido o nível de todos os *switches*, o *Fabric Manager* é utilizado para atribuir o mesmo número de *pod* para os *switches* de borda pertencentes ao mesmo grupo.

2.3 Resumo do Capítulo

Este capítulo apresentou uma análise das principais tecnologias que serviram de base para esta dissertação. As tecnologias que dão suporte à programabilidade na rede, como o protocolo OpenFlow e o controlador NOX, são os pilares para o desenvolvimento da proposta apresentada no próximo capítulo, já que disponibilizam uma certa liberdade para desenvolver e testar, em ambientes reais, novos protocolos e serviços de redes de computadores.

Várias tecnologias abordadas neste capítulo como, por exemplo, a utilização de equipamentos *comoditizados*, rotas na origem, VLB, protocolos de descoberta de topologia, serviços de diretório e topologia são partes importantes da arquitetura SiBF proposta no próximo capítulo.

Capítulo 3

Proposta de Nova Arquitetura de Data Center

Este capítulo apresenta os fundamentos da nova arquitetura SiBF. A proposta apóia-se na argumentação defendida pelo modelo 4D [21], que estabelece a centralização da inteligência de controle da rede através da remoção da operação distribuída presente em vários protocolos como, por exemplo, a resolução de endereço realizada pelo protocolo ARP ou o mecanismo para encaminhamento de pacotes por múltiplos caminhos realizado pelo ECMP (balanceamento de carga). O modelo *Clean Slate 4D* introduz um novo elemento de rede responsável por essas tarefas, o *Decision Element* (elemento de decisão). Além disso, o gerenciamento de redes é dividido em 4 planos: *Decision*, *Dissemination*, *Discovery* e *Data* (decisão, disseminação, descoberta e dados), conforme exemplifica a Figura 3.1. Dessa forma, a tomada de decisão é desviada para um elemento controlador e o plano de dados (encaminhamento) é realizado pelos *switches*. A seguir, descreve-se os requisitos de redes de *data center*, a topologia adotada, os princípios de projeto e os principais blocos funcionais da arquitetura distribuída proposta baseada nos conceitos do modelo 4D.

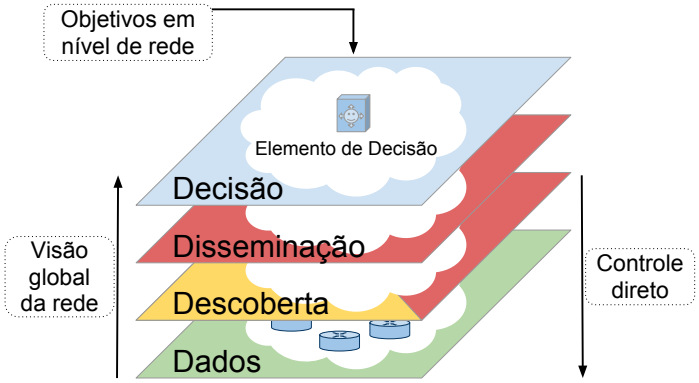


Fig. 3.1: Arquitetura Clean Slate 4D.

3.1 Requisitos de Redes de Data Center

Data centers em grande escala voltados para Internet estão fortalecendo a nova era de computação em nuvem, um paradigma promissor em evolução que pretende disponibilizar capacidade infinita de recursos, com um modelo de serviço onde o usuário paga apenas o que utiliza (colaboração e *pay-as-you-go – utility computing* [5]) e tem um serviço estável e disponível 24 horas, sete dias por semana. O consumidor final não se preocupa com os custos iniciais de implantação, operação e manutenção.

Com o objetivo de acompanhar a crescente demanda e as incertezas quanto ao modelo de negócio, redes de *data centers* em nuvem precisam ser operadas com a relação custo-eficiência e ainda proporcionar um excelente desempenho. Caso contrário, o paradigma *utility computing* pode estagnar. Os esforços atuais em oferecer computação de alta capacidade com baixo custo vão desde programação de aplicações distribuídas (geograficamente) em larga escala e inovação em energia e refrigeração da infraestrutura até repensar a interconexão entre os *racks* de computadores *comoditizados* (servidores). Esta dissertação está direcionada aos problemas de interconexão de computadores em redes de *data center* (DCN) em grande escala. Assim, antes de apresentar a arquitetura SiBF, são discutidas algumas questões relacionadas às exigências de uma rede de *data center*.

Recursos infinitos

Para ser capaz de oferecer a ilusão de um conjunto infinito de recursos de computação disponível sob demanda (*Resource Pool*), é necessário uma infraestrutura com poder de crescimento (*Elastic Computing*), armazenagem de dados e uma ágil interligação de rede. Essas características podem ser obtidas se (i) for possível atribuir os endereços IPs a qualquer máquina virtual em qualquer servidor físico, sem gerar a fragmentação da rede em subcamadas IPs, e (ii) se todos os caminhos da rede estiverem habilitados e com balanceamento de carga eficiente.

Escalabilidade

Um problema fundamental no roteamento de redes, não necessariamente nas redes de *data center*, é a escalabilidade de recursos necessários para realizar a tomada de decisão, que inclui, por exemplo, a obtenção e armazenamento da topologia da rede. A interconexão (dinâmica) de um grande conjunto de endereços IP independentes, na ordem de milhares de máquinas virtuais, requer uma abordagem de encaminhamento Ethernet em larga escala. Porém, as mensagens ARP, o tamanho da tabela de encaminhamento MAC e as limitações do protocolo *Spanning Tree* colocam um limite prático no tamanho da rede.

Desempenho

A largura de banda disponível precisa ser elevada e uniforme em toda a rede e limitada apenas pela taxa máxima de suas interfaces, independente da localização dos nós finais. Isso requer um roteamento livre de congestionamento para qualquer matriz de tráfego, além de um sistema robusto de tolerância a falhas que minimize os impactos provenientes de elementos indisponíveis, como as ligações entre *switches*.

Flexibilidade e capacidade de expansão

A virtualização é o principal responsável pela implementação do modelo *pay-as-you-go*, alocando recursos quando necessário, sem a necessidade de grandes esforços. Com isso, os componentes e princípios adotados para compor a arquitetura do *data center* devem apresentar algumas características, como o suporte a equipamentos legados e capacidade de agregação de novos equipamentos, protocolos e serviços. Dessa forma, uma rede inicialmente pensada da ordem de dezenas de milhares de servidores, quando necessitar expandir para uma ordem de centenas de milhares não precisará, por exemplo, repensar a topologia, modificar os protocolos ou parar o funcionamento atual da rede.

Baixo custo

Uma rede de *data center* é uma grande infraestrutura dotada de comutadores, roteadores, computadores e muitos outros componentes de rede. Portanto, a arquitetura deve sempre aliar desempenho com baixo custo, o que pode ser alcançado, por exemplo, com a adoção de equipamentos *comoditizados* e a remoção de pilha de protocolos que elevam o custo desses equipamentos. Além disso, o consumo de energia é um fator, não só econômico mas ambiental (*Green Network*), que deve ser considerado. Para isso, equipamentos e técnicas podem ser adotados para reduzir o consumo de energia e, consequentemente, reduzir os custos [22].

Suporte a middlebox

Os serviços de *middlebox* (*Firewall*, balanceadores de carga, NAT) normalmente são adicionados no meio do caminho do tráfego da rede de *data center* para atuarem sobre determinados fluxos e com objetivos específicos. As práticas convencionais de gerenciamento de rede (por exemplo, *Spanning Tree Protocol* (STP) e VLAN), forçam o tráfego a atravessar uma sequência de *middleboxes* tornando a configuração geral da rede uma operação difícil e dispendiosa, além de fragmentar os recursos e prejudicar o desempenho [23]. Práticas mais eficazes de suporte a esses componentes são desejáveis e devem ser levadas em consideração na implementação de um *data center*.

Resiliência a falhas

Falhas podem ocorrer, seja por questões lógicas (configuração, sobrecarga, erros, etc) ou físicas (energia, manutenção, desastres, etc). Uma rede de *data center* necessita de um método eficiente para garantir consistência das informações. Por isso, a infraestrutura deve ser capaz de amenizar os impactos causados pelas falhas dos equipamentos de redes (*switches*, servidores, controladores, etc), oferecendo um conjunto de medidas como redundância, detecção, isolamento e recuperação, garantindo uma consistência global da rede, onde o controle logicamente centralizado e as medidas adotadas para tratar as falhas devem ser implementadas em um ambiente mais distribuído possível. O suporte à mobilidade de máquinas virtuais e à distribuição e replicação dos dados são algumas formas de disponibilizar redundância do estado geral da rede para auxiliar na recuperação de falhas.

3.2 Topologia

A partir da definição dos requisitos para o projeto de *data centers*, esta seção inicia a apresentação dos componentes da arquitetura SiBF proposta. A distribuição dos equipamentos de rede em um *data center* é tipicamente organizada de acordo com uma topologia *fat-tree*. A opção por essa topologia deve-se à facilidade de implementação através de *switches* baratos e largamente disponíveis (comoditizados) e por permitir um melhor uso dos recursos da rede [24]. A topologia *fat-tree* adotada (Figura 3.2) possui três camadas:

1. Camada inferior formada por *switches* de topo de *rack* (ToR);
2. Camada intermediária constituída por *switches* de agregação (Aggr);
3. Camada superior de *switches* de núcleo (Core).

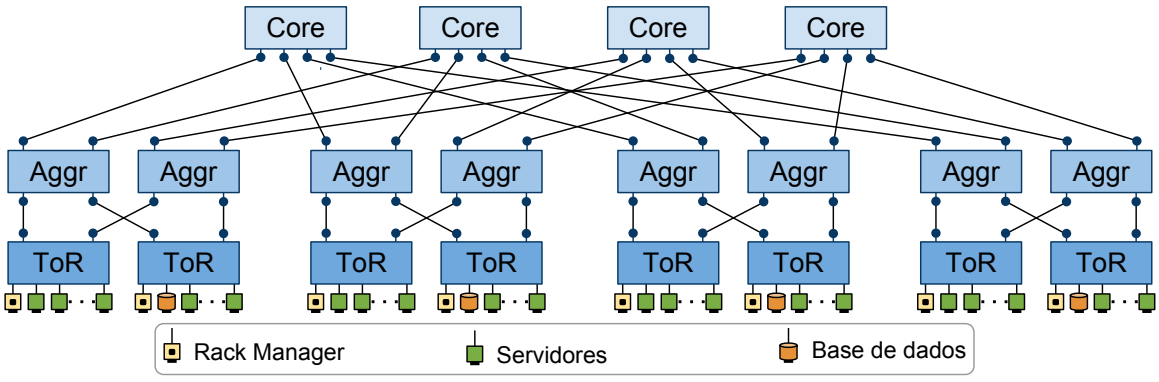


Fig. 3.2: Topologia fat-tree de três camadas.

Essa abordagem proporciona múltiplos caminhos entre qualquer par de servidores, além de facilitar o balanceamento de carga e a tolerância a falhas. Os nós finais conectados aos ToRs podem ser qualquer servidor de aplicação (físico ou virtual), inclusive os que oferecem serviços para suportar o funcionamento da própria infraestrutura como, por exemplo, o *Rack Manager* e o sistema de armazenamento de dados.

3.3 Princípios da Arquitetura

Com base na proposta de arquitetura de *data center*, os princípios de projeto adotados para atender os requisitos de DCN estão representados na Tabela 3.1.

Tab. 3.1: Princípios de projeto adotados para atender os requisitos.

Princípios	Controle logicamente centralizado e fisicamente distribuído						
	Separação identificador/localizador						
	Funcionalidade plug & play						
	Tolerância ampla à falhas						
	Balanceamento de carga						
	Rota na origem						
Requisitos	Baixo custo	X			X		
	Desempenho	X	X	X			X
	Escalabilidade	X	X	X		X	X
	Recursos infinitos				X		
	Resiliência a falha			X			
	Suporte a middlebox				X		
	Flexibilidade e capacidade de expansão				X		

3.3.1 Separação identificador/localizador

A separação semântica entre localizador e identificador propõe criar um novo espaço de nomes (*namespace*), onde o identificador não possui agregação topológica, é único e imutável. Essa abordagem visa, principalmente, reduzir o número de entradas na tabela de roteamento dos *switches*, além de desagregar a semântica presente no protocolo IP. Na arquitetura SiBF, a divisão entre identificador e localizador possui um papel fundamental para viabilizar o compartilhamento dos serviços baseados no endereçamento IP. Este é utilizado apenas para identificar servidores físicos e máquinas virtuais (VMs) dentro do *data center*. Dessa forma, não são impostas restrições de como os endereços

são atribuídos ou utilizados para acesso externo (clientes espalhados na Internet), diferentemente da atribuição de endereços IP baseada na hierarquia de provedores adotada atualmente na Internet. A separação identificador/localizador torna o endereço IP não significativo para o encaminhamento de pacotes dentro da infraestrutura do *data center*. O encaminhamento é realizado no nível da camada 2 (Ethernet), modificada para oferecer um serviço de rota na origem de forma transparente aos nós e aplicações legadas.

3.3.2 Rota na origem

Aproveitando o pequeno diâmetro das topologias de redes de *data center* (*fat-tree*), a abordagem adotada pela arquitetura utiliza o esquema de rota na origem (*strict source routing*). Dessa forma, o encaminhamento nas topologias de DCN em 3 camadas (Figura 3.2) é bastante simplificado, ou seja, qualquer rota entre dois ToRs tem uma trajetória ascendente em direção a um *switch Core* e, em seguida, uma trajetória descendente em direção ao ToR destino, ambas passando por *switches* intermediários (*Aggr*). Essa abordagem permite enviar pacotes por rotas diferentes (balanceamento de carga), além de permitir a seleção da rota menos congestionada (engenharia de tráfego).

A especificação da rota na origem é baseada na utilização do filtro de Bloom nos pacotes (iBF) contendo apenas três elementos identificadores de *switches* $\langle Core, Aggr_{descida}, ToR_{destino} \rangle$. O identificador utilizado neste trabalho corresponde ao endereço MAC do *switch*. O esquema básico adotado para operacionalização do filtro de Bloom consiste na programação do ToR de origem para adicionar, nos campos src-MAC e dst-MAC do cabeçalho do quadro Ethernet, o filtro de Bloom contendo a rota. Na sequência, o ToR encaminha o quadro para o Aggr de subida. Deve ser ressaltado que não há necessidade de incluir o identificador do Aggr de subida no filtro de Bloom já que essa informação é implícita ao ToR. Nos próximos saltos, apenas três encaminhamentos são realizados utilizando a rota na origem baseada no mecanismo iBF, ou seja, no Aggr de subida com o identificador $\langle Core \rangle$, no Core com o identificador $\langle Aggr_{descida} \rangle$ e no Aggr com o identificador $\langle ToR_{destino} \rangle$, conforme exemplificado na Figura 3.3. No ToR de destino, o processo é inverso ao ToR de origem, onde o filtro de Bloom é substituído pelos endereços Ethernet src-MAC (MAC do *switch)* e dst-MAC (MAC do servidor final). Note que o IP não é utilizado no encaminhamento realizado na malha de *switches*, sua utilização é necessária apenas nos ToRs para identificar os servidores.

3.3.3 Balanceamento de carga

Para fornecer uma distribuição do tráfego na rede para qualquer matriz de tráfego, a abordagem adotada é o espalhamento dos fluxos por caminhos aleatórios (*oblivious routing* [25]). O *Rack Manager* é responsável pela seleção aleatória das rotas compactadas nos iBF na direção do ToR de destino.

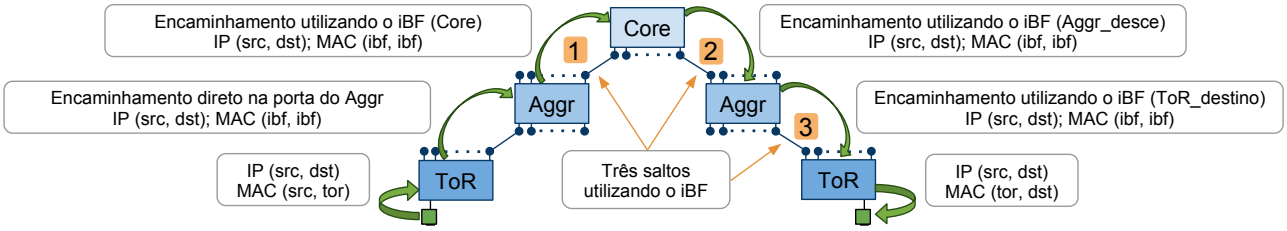


Fig. 3.3: Encaminhamento utilizando iBF.

O RM utiliza a estratégia de balanceamento *valiant load balancing* (VLB como usado no VL2 [3]) que roteia de forma aleatória os fluxos na camada de *switches* intermediários (Aggr e Core). Descarta a necessidade de utilização de protocolos distribuídos nos *switches* ou a adoção de *middleboxes* específicos para o balanceamento, o que, nesses casos, aumentaria o custo de implementação e causaria *overhead*. Políticas de otimização da rede podem ser adotadas (por exemplo, gerenciamento de energia) para reduzir ainda mais os custos, pois o ideal para o balanceamento de cargas é que todos os equipamentos estejam operando (maior número de caminhos disponíveis), mas isto ocasiona um maior consumo de energia.

3.3.4 Funcionalidade plug & play e suporte a servidores e aplicações legadas

O encaminhamento adotado não requer modificações na borda da rede, ou seja, nos nós finais. Servidores legados, interfaces físicas, aplicações e sistemas operacionais são suportados pela arquitetura proposta. Além disso, o comportamento *plug-and-play* do Ethernet é conservado, com autoconfiguração dos nós finais e dos *switches* que fazem parte da solução, não havendo necessidade de qualquer configuração manual desses equipamentos para terem acesso à rede. Sendo assim, diferente de propostas como Monsoon e VL2, não há necessidade de encapsulamento de pacotes em nenhuma fase do encaminhamento, reutilizando os campos de cabeçalhos, semelhante à abordagem adotada no PortLand.

3.3.5 Controle logicamente centralizado e fisicamente distribuído

Atualmente o plano de controle e plano de dados são embutidos nos roteadores e comutadores, através de vários protocolos distribuídos, cada um com uma tarefa específica, tornando esses equipamentos de rede sobrecarregados. Enquanto isso, mais soluções são propostas e mais protocolos são agregados aos equipamentos, aumentando os custos e comprometendo o gerenciamento, o desempenho, a confiabilidade e a escalabilidade da rede. Como descrito no início deste capítulo, a abordagem SiBF adota a proposta incluída no modelo 4D, a qual sugere a separação da tomada de decisão (controle) do encaminhamento (dados) e introduz um novo elemento de rede, o controlador.

Isto implica na "substituição" dos vários protocolos responsáveis pelo encaminhamento por apenas um protocolo que desvie a tomada de decisão para um elemento controlador, que é capaz de realizar o encaminhamento de acordo com as regras definidas nas suas aplicações. Porém, a proposta SiBF estende esse conceito e adota uma abordagem de rede intermediária, ou seja, um controle logicamente centralizado, onde apenas o estado da topologia e do diretório de servidores são mantidos globalmente (visão global da rede), mas com controladores distribuídos fisicamente para atuação sobre um número limitado de *switches*. Esses controladores são denominados de *Rack Managers* (Seção 4.1.2) e são responsáveis pela gerência dos *switches* com o protocolo OpenFlow habilitado. Com isso, o controle e os protocolos são removidos dos roteadores e comutadores, tornando-os mais em conta financeiramente.

3.3.6 Tolerância ampla à falhas

A premissa de que qualquer elemento pode falhar é adotada na concepção e implementação da arquitetura. Em uma infraestrutura com dezenas de milhares de elementos, falhas são comuns e constantes e podem afetar o desempenho da rede. Considerando um *data center* com 50.000 elementos (servidores e *switches*), e com um excelente tempo médio entre falhas de 10 anos para cada elemento, há de se esperar que todo dia tenha, pelo menos, um elemento em mau funcionamento. Nesse contexto, o desenho da arquitetura e os serviços de rede devem ser à prova de falhas (*design for failure*), assumindo que qualquer componente pode falhar a qualquer momento.

3.4 Serviço de Encaminhamento com Filtro de Bloom Livre de Falsos Positivos

Devido às suas propriedades, a adoção de filtros de Bloom para o encaminhamento de pacotes pode ocasionar falsos positivos com uma certa probabilidade. Essa característica ocasiona alguns problemas que, dependendo do resultado da verificação do filtro de Bloom nos *switches*, pode inviabilizar o encaminhamento do pacote. A seguir, são apresentados o problema e a solução adotada para remover os possíveis falsos positivos.

3.4.1 Filtro de Bloom

Um filtro de Bloom [26] é uma estrutura de dados que identifica se determinado elemento está ou não presente nessa estrutura. É implementado como um vetor de m bits configurados inicialmente para zero $bf = \{b_1, b_2, ..., b_m = 0\}$ (cf. Figura 3.4), com um conjunto $S = \{x_1, x_2, ..., x_n\}$ de

n elementos inseridos por k funções de *hash* independentes $h_1, h_2, ..., h_k$ com intervalo de valores $1 \leq h(x) \leq m$. Cada elemento, ao ser inserido no filtro, passa por k funções de *hash* cuja ação consiste na atribuição do valor 1 ao bit na posição do vetor associada ao resultado do *hashing*. Para verificar se algum elemento está presente, os resultados de todas as k funções de *hash* devem possuir o valor 1 no vetor. Basta um único dos bits anteriores apresentar o valor zero para concluir que o elemento não está presente. Nesse caso, não há possibilidade da ocorrência de falsos negativos. Um falso positivo ocorre quando todos os resultados das k funções de *hash* em um elemento que não foi inserido apresentam o valor 1. A Figura 3.5 exemplifica um Filtro de Bloom para $m = 32$, $k = 3$ e $n = 3$, onde um elemento qualquer (y) é verificado se está ou não presente na estrutura.

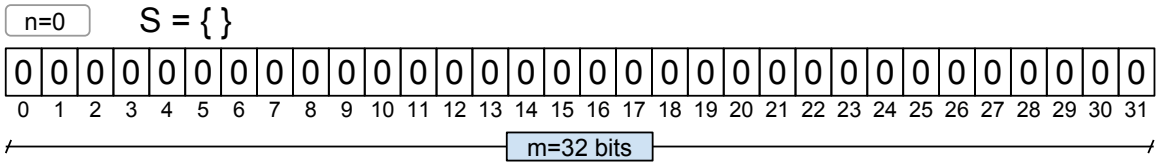


Fig. 3.4: Filtro de Bloom vazio.

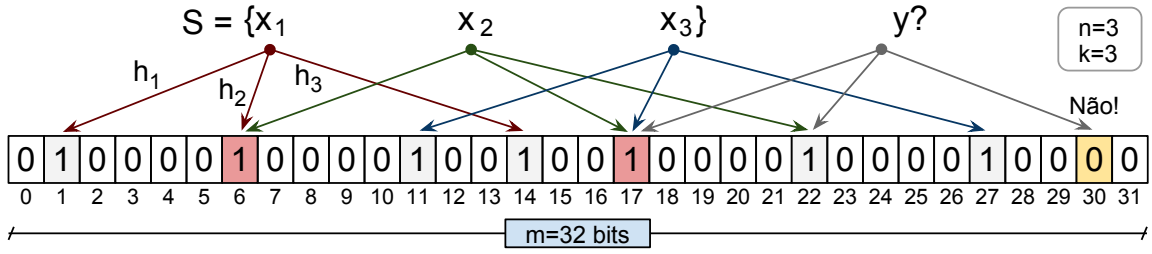


Fig. 3.5: Filtro de Bloom com $m = 32$, $k = 3$ e $n = 3$.

A estimativa normalmente utilizada para a probabilidade de falso positivo de um filtro de Bloom de tamanho m, com n elementos inseridos e com número de k funções *hash* é expressa pela função [27]:

$$p^k = \left[1 - \left(1 - \frac{1}{m}\right)^{k*n}\right]^k \tag{3.1}$$

Dada a taxa alta de bits por elemento na arquitetura SiBF ($m/n = 32$, para $m = 96$ e $n = 3$), a probabilidade de falsos positivos é baixa, ou seja, após a escolha de um número k ótimo de funções de *hash*, a taxa estimada de falsos positivos dada por p^k é na ordem de 10^{-7}.

3.4.2 Impacto de falsos positivos no encaminhamento

O serviço de topologia, após a descoberta da topologia, instala na tabela de encaminhamento do *switch* uma entrada de fluxo correspondente a cada interface de saída do *switch*. Cada uma dessas entradas equivale a um filtro de Bloom codificado com a identificação do vizinho detectado.

Quando um pacote com iBF chega ao *switch*, a máscara de bits é comparada com as entradas na tabela. Essa comparação é realizada com base em prioridades, onde uma entrada com maior correspondência exata nos campos de pacotes utilizados pelo OpenFlow, ou seja, número maior de combinação nos campos do pacote (*inport*, *Ether src*, *Ether dst*, ...), possui a maior prioridade. Se uma entrada for encontrada, o pacote é enviado à porta associada, conforme ilustra a Figura 3.6. Um falso positivo ocasiona o retorno de duas (ou mais) entradas na tabela de encaminhamento, sendo que uma das entradas deverá ser escolhida, dependendo dos critérios de prioridade, onde em último caso, se todas as entradas possuem a mesma prioridade, o *switch* é livre para selecionar qualquer uma das correspondências, podendo resultar no envio do pacote para o vizinho errado.

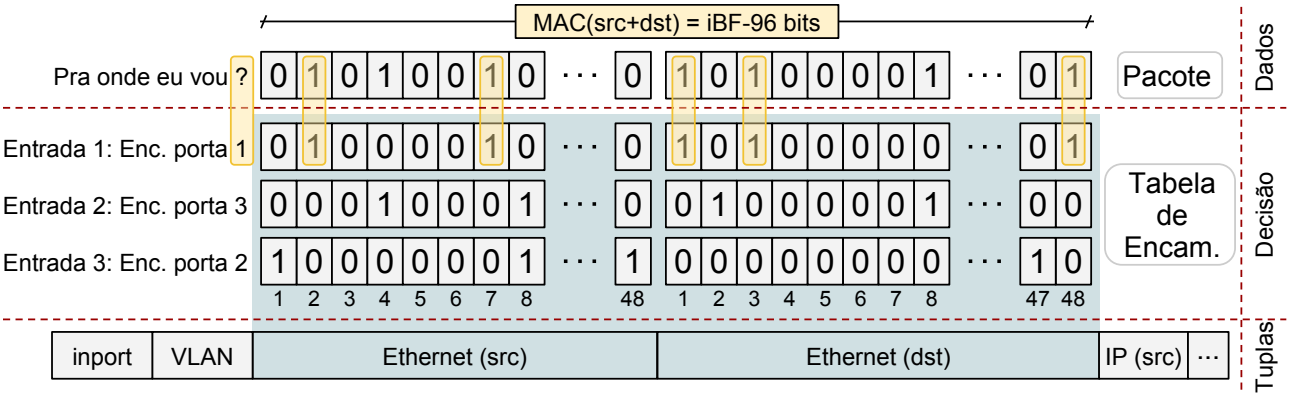


Fig. 3.6: Tomada de decisão nos *switches* quando chega um pacote com iBF.

Apesar de baixa, a probabilidade de falsos positivos existe e por isso, deve ser levada em consideração na implementação da arquitetura. Uma opção para contornar a ocorrência de falsos positivos na tomada de decisão nos *switches* é a realização do *multicast*. Nesse caso, o pacote será encaminhado pela porta correta, na qual a rota foi gerada pelo ToR de origem, e por mais uma (ou mais) porta(s) e, no próximo *switch*, com alta probabilidade, será descartado ao carecer de uma entrada válida. Porém, em função da topologia, essa abordagem pode ocasionar *loops*.

Outra opção consiste no encaminhamento *unicast*. Nessa opção, se a escolha por um dos caminhos não for a opção correta, o pacote será encaminhado pelo próximo *switch* (que não terá uma entrada válida para encaminhar aquele pacote) ao controlador permitindo, nesse caso, que uma nova rota seja definida. Mas essa abordagem ocasiona *overhead* extra em tempo de encaminhamento.

3.4.3 Remoção dos falsos positivos

Ao longo da implementação do encaminhamento baseado em filtro de Bloom nos pacotes, através da utilização do OpenFlow, foram surgindo problemas e estratégias referentes à remoção dos possíveis falsos positivos.

Em primeiro lugar, a estratégia adotada neste trabalho para a remoção dos possíveis falsos positivos explora o conceito do poder de escolha através de duas opções: (1) múltiplos caminhos e (2) múltiplas representações dos iBF, ou seja, são calculados os iBFs para múltiplos caminhos disponibilizados pela topologia *fat-tree* e para cada caminho são gerados candidatos adicionais usando diferentes conjuntos de funções *hash* (similar ao *Link ID Tags*, proposto em [28]). Utilizando as informações da topologia, o serviço de encaminhamento pode facilmente verificar com antecedência se algum candidato pode causar falsos positivos ao longo do caminho. Dessa forma, os candidatos selecionados aleatoriamente são descartados na geração da rota na origem.

Em segundo lugar, em trabalhos futuros, podem ser instaladas regras simples nos switches Aggr, onde os pacotes nunca são encaminhados na mesma direção em que chegaram (se o pacote veio de um ToR é enviado para um Core, e vice-versa), exceto nos casos onde os ToR de origem e destino encontram-se conectados no mesmo Aggr, não necessitando o pacote subir até um Core, por consequência, não há necessidade de um identificador de Core no iBF ($n = 2$).

Em terceiro lugar, dada a topologia *fat-tree* bem definida, em alguns casos os falsos positivos são resolvidos automaticamente por força da topologia. É o caso onde um falso positivo ocorre no caminho de subida em um Aggr (Aggr encaminha para o Core). Nesse caso, o Aggr encaminha para o Core errado. Na topologia *fat-tree* adotada, cada Core conecta-se com todos os Aggr, desta forma o Core corrige o erro (a menos que tenha outro falso positivo) encaminhando o pacote para o Aggr de descida, uma vez que terá a entrada correspondente no iBF para este Aggr.

Outras estratégias para lidar com os possíveis falsos positivos referem-se especificamente à tecnologia OpenFlow, como utilizar o esquema de prioridades do protocolo OpenFlow para diferentes fluxos. Essas técnicas não foram adotadas por hora, mas podem ser em trabalhos futuros, para dar suporte a encaminhamento *multicast* baseado no iBF, por exemplo. Dessa forma, abaixo são descritos o comportamento para potencias falsos positivos nos *switches*:

- (i) No Aggr de subida (encaminhamento para o Core), falsos positivos são automaticamente corrigidos, com alta probabilidade e sem salto adicional (sem aumentar o *stretch*), pois o Core possui uma entrada válida para o próximo Aggr de descida.

- (ii) No Core (encaminhamento para o Aggr de descida), falsos positivos resultam no Aggr receber um pacote que não tem uma entrada válida na tabela de encaminhamento. Nesse caso, o OpenFlow especifica que o pacote deve ser enviado para o controlador.

- (iii) No Aggr de descida (encaminhamento para o ToR destino), este caso é o mesmo do anterior. O próximo salto vai para o ToR selecionado erradamente, onde esse ToR não terá entrada válida para entregar o pacote para IP destino, enviando-o para o controlador.

Em qualquer dos casos de falsos positivos descritos acima, o primeiro pacote de um fluxo será encaminhado para o controlador, que calcula um caminho alternativo e instala as entradas necessárias para corrigir temporariamente o problema, marcando o iBF específico que representa esse caminho como inválido para futuras utilizações. Pacotes tratados dessa maneira não serão perdidos, mas sofrerão um pequeno atraso até que o controlador instale as entradas de fluxos necessárias (haverá saltos adicionais). Como o controlador possui a visão global da topologia e existem múltiplos caminhos, o RM é capaz de pré-calcular e manter uma matriz de ToR de origem e destino preenchida apenas com iBFs livres de falsos positivos.

3.5 Serviços de Topologia e Diretório

Assim como outras propostas de novas arquiteturas de *data center* (por exemplo, VL2 [3], Monsoon [20] e PortLand [4]), a arquitetura apresentada neste trabalho propõe um serviço escalável de diretório para manter o mapeamento entre o identificador IP do servidor (ou máquina virtual) e o *switch* de borda ToR onde o nó final está ligado. O serviço de topologia é o resultado de um protocolo de descoberta de topologia baseado em uma extensão do *Link Layer Discovery Protocol* (LLDP). Ambos os serviços (topologia e diretório) fornecem as informações globais necessárias às funções de controle direto. O gerenciamento do encaminhamento dos fluxos no *data center* tolera um certo grau de inconsistência nas informações fornecidas pelos serviços de topologia e diretório. Essa flexibilidade permite a implementação desses serviços através de um sistema distribuído escalável, confiável e de alto desempenho, do mesmo modo como em muitas outras aplicações na nuvem. Os serviços de topologia e diretório ao disponibilizarem uma base de dados única e global ao controlador *Rack Manager*, estão fornecendo um controle logicamente centralizado, pois cada RM trabalha de forma independente mantendo o número de requisições por segundo (eventos) sob controle, acreditando ser o único controlador na rede, sendo que na verdade ele atua como controlador em um escopo limitado. Porém, quando se observa o escopo global da rede, o controle está totalmente distribuído nas várias instâncias de *Rack Managers*.

3.5.1 Protocolo de descoberta

O conhecimento da topologia é um pré-requisito para o encaminhamento utilizando rotas na origem. Um ponto não trivial é a inferência correta da topologia da árvore e o papel que cada *switch* de-

sempenha na rede (ou seja, ToR, Core ou Aggr). Essa questão é mais crítica no tempo de inicialização dos *switches*, já que uma das exigências é suportar o comportamento *plug & play* do Ethernet, evitando qualquer intervenção manual (por exemplo, configurar via *script* o papel de cada *switch*). Essa característica evita não apenas diminuir os esforços operacionais, como na substituição de *switches* em mal funcionamento, mas também é fundamental para o correto (e otimizado) encaminhamento de pacotes utilizando o iBF. Para isso, foi desenvolvido um algoritmo (ver algoritmo da Figura 4.3) que automatiza essa inferência. O *Tree and Role Discovery Protocol* utiliza uma extensão *Type-length-value* (TLV) no LLDP para divulgar a identificação de cada *switch*. Essa informação é propagada entre os vizinhos, onde cada um descobre qual é o seu papel na topologia de acordo com as informações recebidas dos seus vizinhos diretos ou dos nós finais. Um problema semelhante é abordado em PortLand [4], onde os *switches* exigem a descoberta de sua localização específica dentro da hierarquia na topologia, para compor seu endereço PMAC na forma *pod.position.port.vmid*. Nosso protocolo é bastante simples e exige apenas que se identifique a camada na qual o *switch* está inserido, o que representa o seu papel dentro da rede como ToR, Aggr ou Core. Mais detalhes do protocolo estão disponíveis na seção de Implementação (4.1.2).

3.5.2 Serviço de topologia

Além de auxiliar no tratamento de falhas, o protocolo de descoberta fornece as informações de topologia necessárias para o encaminhamento de pacotes utilizando rotas na origem. Com isso, o *Topology Service* (TS) deve manter atualizado o estado global da topologia da rede procurando oferecer a maior consistência possível para as informações. A descoberta da topologia realiza a identificação dos *switches* (Core, Aggr e ToR), associa o vizinho descoberto a uma interface de saída (porta) e instala as entradas nos *switches* Aggr e Core contendo o filtro de Bloom com a identificação do vizinho. As informações da topologia, descobertas através de cada TS, são inseridas na base de dados por quaisquer dos *Traffic Matrix*s (TMs) responsáveis pelo controle dos *switches* e pela implementação distribuída do protocolo de descoberta. O TS também é responsável pela recuperação da topologia (global) e por disponibilizá-la como um serviço ao RM.

3.5.3 Serviço de diretório

Com a separação identificador e localizador, o *Rack Manager* necessita conhecer previamente o ToR em que o IP (identificador) de destino está conectado. Como um dos objetivos da arquitetura é a escalabilidade, a propagação de pacotes ARP através de *broadcast* é eliminada. Portanto, o *Directory Service* (DS) deve manter um mapeamento entre IP e ToR. Essa informação é mesclada na base de dados, e o próprio DS recupera os mapeamentos dos outros ToRs permitindo-o ter uma visão do

estado global. Outro mapeamento, que não é requisito de estado global, mas único de cada ToR, é a associação do IP e MAC à porta de saída do ToR. Por questões de recuperação de falhas do Rack Manager, esse mapeamento também é armazenando na base de dados.

3.5.4 Tecnologia de implementação do TS/DS

A arquitetura proposta no trabalho adota um sistema de base de dados distribuída baseada no par <chave, valor> do tipo *Not only SQL* (NoSQL) [29]. As implementações nos *cloud data center* que usam uma estrutura desse tipo apresentam APIs de fácil utilização e atendem os requisitos identificados (consistência, escalabilidade, confiabilidade). Bancos de dados relacionais apresentam custos elevados e não se justificam quando não se requer garantias estritas de ACID (atomicidade, consistência, isolamento e durabilidade). As bases de dados não relacionais disponíveis são distribuídas e apresentam, geralmente, uma eficiente replicação dos dados, formando uma base para a implementação de serviços de apoio ao desenvolvimento de aplicações. Tipicamente, a API oferece operações simples (`get(key)`, `set(key, value)`, `remove(key)`) e a sua implementação e funcionamento internos (replicação, tolerância a falhas, consistência, versionamento) são transparentes ao desenvolvedor. Dentre os vários exemplos desse tipo de bases de dados podemos citar o Dynamo da Amazon [15] e o Keyspace [30] (adotado neste trabalho).

3.6 Resumo do Capítulo

O objetivo principal deste capítulo foi expor os métodos adotados na concepção da arquitetura SiBF. Os requisitos de um *data center* delimitaram os problemas ou metas a serem abrangidas pela solução proposta, que vai desde a topologia adotada, neste caso uma *fat tree* em três camadas, até os princípios da arquitetura que dão base à implementação, como a separação identificador e localizador, um tema bastante abordado em outras literaturas como em *Routing on Flat Labels* (ROFL) [31] e *Host Identity Indirection Infrastructure* (Hi3) [32]. Outra questão importante relaciona-se aos falsos positivos já que a ocorrência destes pode comprometer o serviço de encaminhamento. Além disso, os serviços TS e DS garantem a expansão da rede (escala) e servem de base ao argumento de que o paradigma de controle centralizado pode ser fisicamente distribuído conforme discutido em Onix [12].

Capítulo 4

Implementação e Validação

Este capítulo apresenta as tecnologias e métodos usados para implementação e validação da proposta e encontra-se dividido em duas seções: na Seção 4.1 são apresentadas informações a respeito da execução, efetivação e/ou modificação das tecnologias adotadas na concepção da arquitetura SiBF; na Seção 4.2 são discutidos os resultados dos testes relativos à implementação do protótipo.

4.1 Implementação

As particularidades quanto ao protocolo OpenFlow (para realizar o encaminhamento com o iBF), o controlador NOX (para atuar como gerenciador dos *switches* OpenFlow) e a base de dados (para fornecer um estado único das informações de diretório e topologia), assim como os componentes do ambiente de testes e a sequência de mensagens dentro desse ambiente, são detalhadas a seguir.

4.1.1 Protocolo OpenFlow

A principal característica do OpenFlow consiste na definição de uma tabela de fluxos cujas entradas contêm um conjunto de campos do cabeçalho de pacote. Esse conjunto é composto por uma tupla formada por 12 elementos (versão 1.0) e por uma lista de ações suportadas via *hardware* como, por exemplo, o envio de um fluxo para uma determinada porta, o encapsulamento e transmissão do pacote para o controlador ou, ainda, o descarte do pacote. Para viabilizar a implementação do protótipo, e afim de permitir o encaminhamento com base na codificação do filtro de Bloom incluída nos campos de endereço MAC do quadro Ethernet, foi introduzida uma pequena alteração na implementação do protocolo OpenFlow (v. 0.89rev2 e v 1.0). O encaminhamento baseado no iBF utiliza uma máscara de bits com apenas k bits definidos para 1, com isso, é necessário adicionar o suporte a esse comportamento especial na implementação do caminho de dados (*datapath*) do OpenFlow. Felizmente, são

necessárias apenas a (i) inserção de uma função[1] e (ii) a mudança de duas linhas do código[2] original da função que verifica a igualdade entre o fluxo e as regras instaladas nas tabelas dos *switches*. A Figura 4.1 mostra a tupla de campos de cabeçalho disponível pelo OpenFlow e identifica os campos que são utilizados pela arquitetura, no caso, os dois campos de endereço Ethernet (`src` e `dst`) para inclusão do iBF de 96 bits, e o campo relacionado ao IP, o qual é interpretado na arquitetura como identificador do nó final. O campo VLAN será explicado mais adiante.

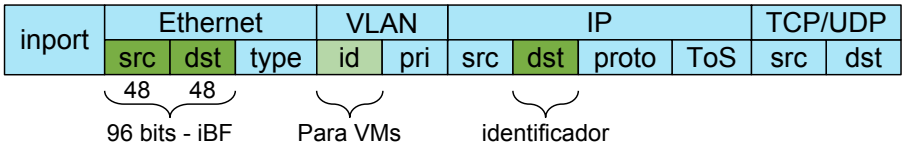


Fig. 4.1: Tuplas disponíveis pelo OpenFlow e as 3 utilizadas.

4.1.2 Gerenciador de Rack

O Gerenciador de *Rack*, ou *Rack Manager* (RM), atua como um controlador de *switches* e a sua implementação é instanciada como um conjunto de componentes que executam no contexto do controlador NOX. Para realizar o encaminhamento com o iBF e fornecer os serviços de topologia e diretório, foi necessário o desenvolvimento de alguns componentes adicionais para o NOX (componentes do usuário na Figura 4.2).

Rack Manager Core

O *Rack Manager Core* (RMC) é o principal componente do Gerenciador de *Rack* e responsável pelo gerenciamento dos novos fluxos que chegam ao *switch* e a configuração desses fluxos através dos serviços de diretório e topologia. O RMC mantém uma base de dados em *cache* (compartilhada dos componentes TS e DS) com informações dos servidores descobertos em cada ToR e um mapeamento entre IP, MAC e porta do ToR no qual esses servidores estão anexados. Essas informações são atualizadas no *Directory Service*. O RMC interage periodicamente com os componentes *Topology Service* e o *Directory Service* para obter a topologia completa e as informações globais de diretório de servidores. Dessa forma, o RMC constrói uma matriz com os candidatos iBFs livres de falsos positivos para cada caminho entre qualquer combinação de pares de ToRs.

[1] Função `flow_fields_iBF`

[2] Função `flow_fields_match` em `openflow1.0.0/udatapath/switchflow.c` ou `openflow-0.9.0/datapath/flow.c`

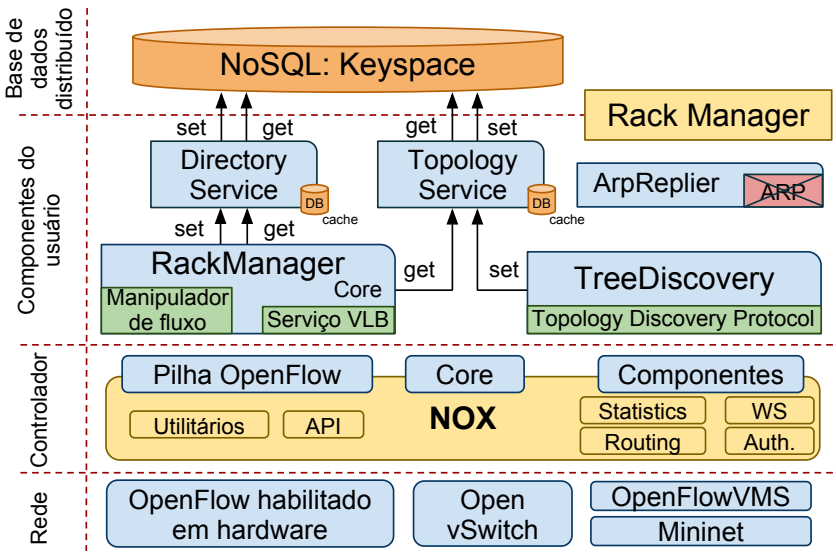


Fig. 4.2: Arquitetura do SiBF.

Tree Discovery

O componente *Tree Discovery* implementa o algoritmo do protocolo de descoberta da Figura 4.3, instala regras nos *switches* OpenFlow e atualiza a base de dados através do *Topology Service*. A transição de estado dos *switches*, resultado da atuação do protocolo, está representada na Figura 4.4. Basicamente, quando um *switch* se junta à rede (`switch_join`), o *Tree Discovery* cria um pacote LLDP com a identificação (`Role`) configurada como indefinida (`Undefined`), representada pelo valor zero na variável TLV ($id = 0$, Figura 4.4(a)), e envia para todas as portas do *switch* (`SendAllPorts`), com frequência de reenvio definida por uma constante de tempo. A partir da troca de pacotes LLDP, as ligações entre os vizinhos são criadas. Cada par de *switches* cria duas ligações entre si, que são diferentes e independentes, e são armazenadas na forma de uma tupla de 4 elementos (`dp_id`, `inport`, `chassid`, `portid`):

- **dp_id**: Identificação do *switch*;
- **inport**: Porta em que o pacote LLDP foi recebido;
- **chassid**: Identificação do *switch* vizinho;
- **portid**: Porta do *switch* vizinho ao qual o *switch* está conectado.

Quando um *switch* de borda (ToR) recebe um pacote ARP (função `arp_receive_server` do algoritmo) representado na Figura 4.4(b), sua identificação é alterada para `tor` e a TLV passa a ser enviada com valor um ($id = 1$, Figura 4.4(c)), desencadeando a descoberta da topologia. A

identificação $id = 1$ tem prioridade sobre as outras, ou seja, uma vez que um *switch* tenha sido configurado como ToR, independente dos vizinhos, ele sempre será ToR.

início switch_join
  Role ← Undefined;
  `SendAllPorts(`*llpd, Role*`)`;
fim

início arp_receive_server
  se *Role* ! = *ToR* **então**
    Role ← ToR;
  fim
fim

início lldp_receive_neighbors
  NbRole ← neighbors.Role;
  se *NbRole* = *(Core* ***or*** *ToR)* **então**
    Role ← Aggr;
  senão se *NbRole* = *Aggr* **então**
    Role ← Core;
  fim
fim

Fig. 4.3: Algoritmo (Tree and Role Discovery Protocol).

Um *switch*, ao receber notificações (`lldp_receive_neighbors`) de pelo menos um ToR ou um Core, altera sua identificação para `aggr` e passa a enviar os pacotes LLDPs com TLV configurado para dois ($id = 2$, Figura 4.4(d)). Da mesma forma, um *switch* altera sua identificação para `core` ao receber a notificação de um vizinho Aggr passando a enviar TLV com valor três ($id = 3$, Figura 4.4(e)). Nesse último procedimento, nota-se que o *switch* de borda que não recebeu pacotes ARP de nenhum nó, ao receber pacotes LLDPs com TLV igual a 2 altera de forma "incorreta" sua identificação para Core. Porém, esta momentânea identificação errada não prejudica o encaminhamento de pacotes entre dois nós que estão abaixo de ToRs devidamente identificados. Como o identificador de Core é o menos prioritário, não considerando o zero ($id = 0$), o *switch* com identificação Core, ao receber um pacote ARP, altera sua identificação para ToR e atualiza o identificador TLV para um ($id = 1$), conforme Figura 4.4(f), finalizando os possíveis estados em que um *switch* pode assumir no processso de descoberta.

Este mapeamento, entre identificação do *switch* e o papel que ele desempenha na rede, é enviado para o Serviço de Topologia para ser armazenado na base de dados. Após identificar o papel de cada *switch*, o *Tree Discovery* insere regras permanentes de encaminhamento iBF apenas nos *switches* Core e Aggr.

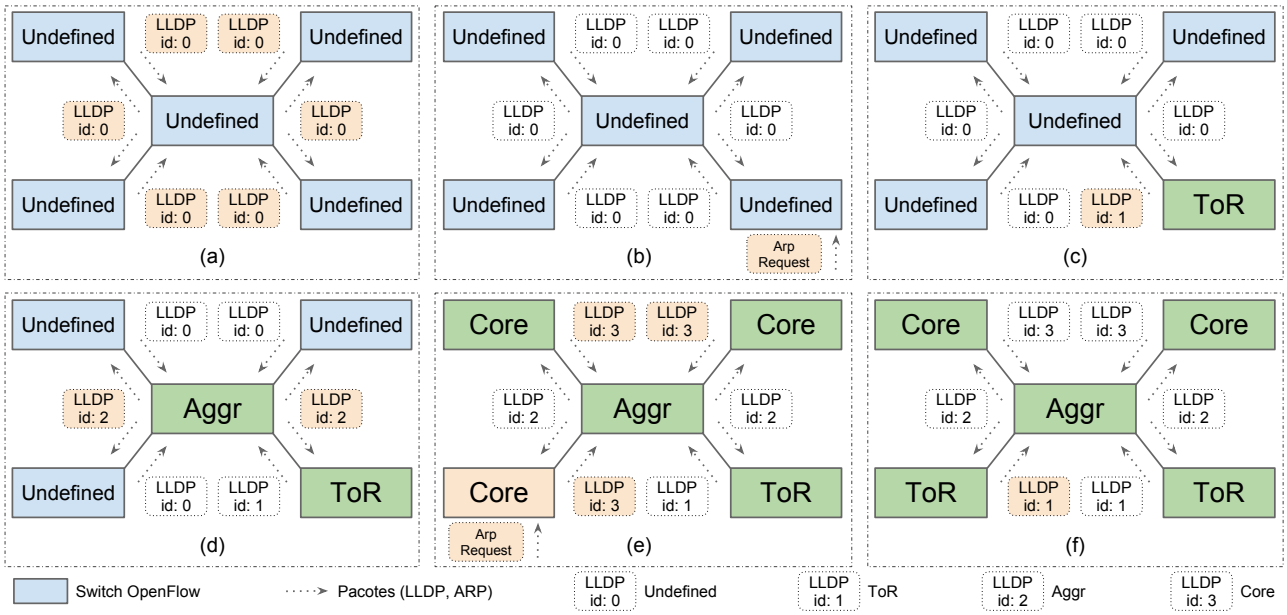


Fig. 4.4: Transição de estado do protocolo de descoberta.

Arp Replier

Possui a tarefa de registrar e responder aos pacotes ARP que chegam aos *switches* de borda, com isso, elimina a inundação na rede (*flooding*) ocasionada pela propagação de pacotes ARP, mas a pilha de protocolos nos nós finais é mantida inalterada. Para qualquer requisição de ARP, o *Arp Replier* inclui na resposta, no campo de MAC destino, o endereço MAC associado ao ToR, na verdade um MAC "falso". Isso garante que a tabela ARP do servidor (físico ou virtual) tenha uma entrada, mesmo que única, para cada IP de destino, onde o ToR acaba atuando como se fosse um *gateway* de IP. Esta implementação, a remoção do *broadcast* do protocolo ARP na malha de *switches*, é fundamental em redes de grande escala, como visto nas propostas VL2, PortLand e Monsoon. A diferença básica entre o *Arp Replier* e o Monsoon, por exemplo, é que o *Agent Monsoon* é implementado na pilha de protocolo dos servidores, o que não ocorre com o *Arp Replier*.

Topology Service e Directory Service:

As implementações dos componentes DS e TS seguem uma mesma metodologia. Oferecem métodos de inserção (*set*) e recuperação (*get*) das informações na base de dados. Os métodos de inserção recebem apenas uma chave e um valor. Cada método é responsável por codificar a chave em uma outra chave para armazenamento na base de dados composta pela chave e prefixo (`set("pre+key", value)`). Esse procedimento é necessário pois os dados encontram-se em um único domínio e, nesse caso, o prefixo é utilizado para distinguir os tipos de informações (por exemplo, lista de Core, mapea-

mento IP/ToR). Quando uma informação é solicitada, um conjunto de valores é retornado baseado na chave e no prefixo associados. Essa decisão de implementação deve-se às características do esquema de distribuição, já que cada RMC armazena na base de dados apenas as suas informações locais, mas necessita obter as informações de toda a topologia. Quando o RMC precisa realizar uma consulta, tanto na lista de diretório quanto na de topologia, essas não são acessadas diretamente na base de dados, o que geraria um *overhead* significativo. O TS e o DS são responsáveis apenas pela atualização da base local (*cache*) do RMC. Dessa forma, temos uma base de dados logicamente centralizada (mas distribuída fisicamente) e replicada em cada RMC.

O DS realiza dois mapeamentos associados aos nós finais (servidores físicos e VMs) e aos *switches* de borda: 1) armazenamento da tupla IP, MAC e porta; 2) armazenamento da tupla IP e ToR. O primeiro mapeamento é utilizado pelo RMC do ToR (destino) para entregar o pacote ao servidor, realizando a associação entre o <IP, MAC> e <MAC, porta>. O segundo mapeamento é utilizado pelo RMC do ToR (origem) para localizar o ToR responsável pelo IP_{dst}. No caso de máquinas virtuais endereçadas com IPs privados, adiciona-se o identificador da VLAN (ver Figura 4.1) ou da aplicação responsável pelas VMs para garantir um mapeamento único das instâncias virtuais nos servidores físicos. O TS é responsável pela manipulação de quatro dicionários, onde três armazenam as informações da topologia, um para cada nível na árvore *fat-tree*, ou seja, ToR, Aggr e Core. O quarto contém as identificações internas dos *switches* utilizadas pelo protocolo de descoberta para criar as ligações (*links*) entre os vizinhos.

4.1.3 NoSQL Keyspace

A escolha pelo sistema distribuído de armazenamento do par <chave,valor> baseado no Keyspace [30] deve-se à fácil implementação na infraestrutura da nuvem. Além de ser disponibilizado em código aberto e escrito em Python, o que facilita a integração com o código do NOX, o Keyspace oferece consistência, tolerância a falhas e alta disponibilidade, conforme características listadas na Figura 4.5. O algoritmo distribuído Paxos (algoritmo de consenso) é utilizado para manter a consistência dos dados replicados (chave/valores), onde um quorum de nós verifica a consistência desses dados. Com isso, pelo menos dois nós devem estar ativos para o Keyspace garantir a consistência das informações. No ambiente do *cloud data center*, o número de nós do Keyspace pode ser tão grande quanto necessário, envolvendo componentes da própria infraestrutura como, por exemplo, servidores ou *Rack Managers*. No ambiente de teste são utilizados 4 nós e na implementação utilizou-se uma API em Python integrada aos componentes do NOX (TS e DS). Exemplos destacáveis de sistemas NoSQL usados comumente nos *clusters* dos *cloud data centers* incluem o Hadoop Distributed File System (usado pelo Yahoo!), o Dynamite (baseado no Dynamo do Amazon), o Cassandra (usado pelo Facebook) e os populares MongoDB e CouchDB.

Fig. 4.5: Características do NoSQL Keyspace.

4.1.4 Sequência de Mensagens

O diagrama de fluxo de pacotes da Figura 4.6 mostra como as comunicações ocorrem na implementação do protótipo (no exemplo, uma conexão TCP), onde as setas tracejadas, setas contínuas e linha tracejada representam, respectivamente, pacotes de dados, mensagens do protocolo OpenFlow e fluxo de dados após as instalações das regras nos *switches*. A atividade na rede começa quando um servidor envia uma solicitação ARP (*Resquest*) para algum IP destino (Etapa 0.a), por exemplo, para resolver o endereço do servidor DNS. A solicitação alcança o *switch* ToR, que não possui nenhuma entrada correspondente na tabela de fluxo e, então, armazena o pacote em memória (*buffer*) e informa o controlador (NOX/RM) para tomar uma decisão sobre o novo fluxo. No controlador, o evento `packet-in` é passado para todos os módulos que têm interesse em pacotes ARP (*Rack Manager*, *Tree Discovery* e *Arp Replier*) sendo que cada módulo toma as suas decisões de forma independente.

O RM descobre a localização do servidor e o associa à porta de saída do *switch* que desencadeou o evento (Ação A). Em seguida, envia um comando (`flow-mod`) para instalar uma entrada semi-permanente na tabela de fluxos do *switch* (que expira em 180 segundos de inatividade na rede) para futuros pacotes com IP destino igual ao do servidor e as ações associadas definidas como: (i) reescrever os campos de cabeçalho Ethernet (MAC src e dst) com um falso MAC (no campo MAC_{src}) e o MAC original do servidor (no campo MAC_{dst}); (ii) encaminhar o pacote para a porta onde o servidor encontra-se ligado. O *Tree Discovery* identifica o *switch* como um ToR e atualiza o estado do Serviço de Topologia (Ação T). O *Arp Replier* responde com um falso ARP (*Reply*) contendo a identificação do ToR (Ação R, Etapa 0.b).

Na Etapa 1, o servidor envia um ARP *Request* para um IP destino assumindo que a Etapa 0 já tenha sido realizada e o ToR já disponibiliza os IPs (src e dst) no seu Serviço de Diretório (Ação B).

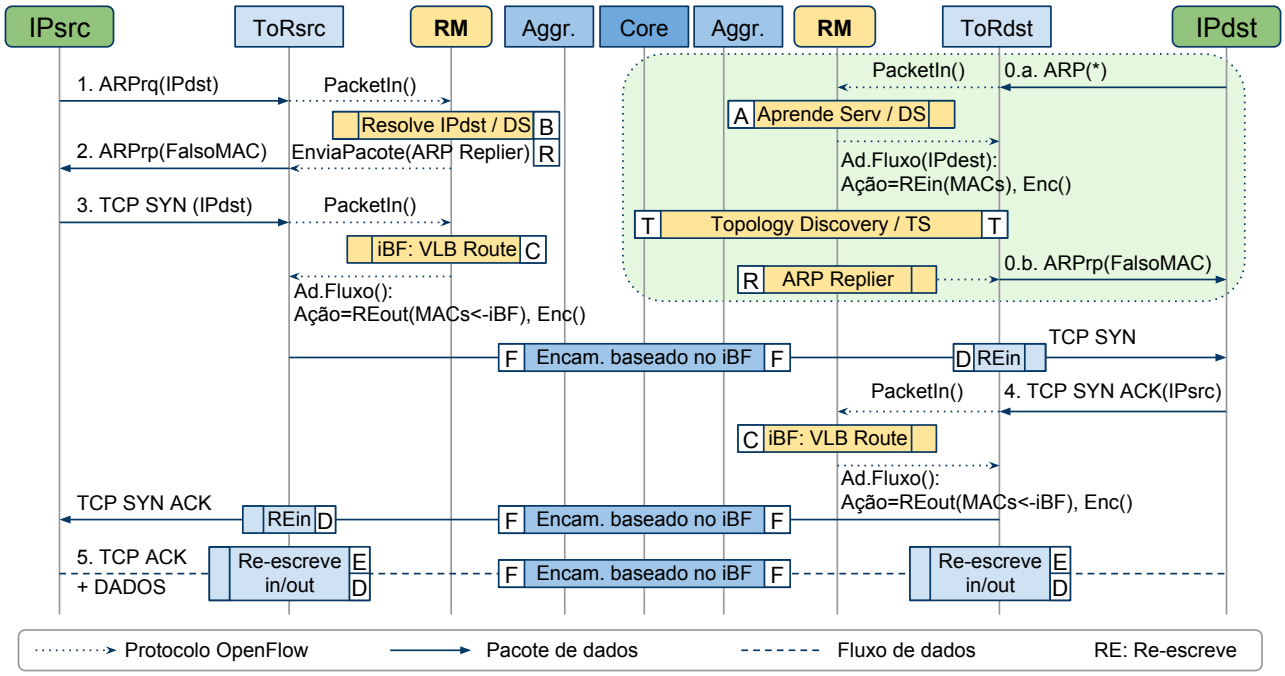


Fig. 4.6: Sequência de fluxo de pacotes em uma instanciação do OpenFlow baseado no SiBF.

Depois de receber o ARP *reply* (Ação R, Etapa 2) com o falso MAC (identificador do ToR), o servidor de origem envia um pacote TCP SYN que alcança o *switch* ToR, o qual encaminha o pacote para o controlador (Etapa 3). O RM escolhe aleatoriamente (VLB) o iBF para o ToR destino (Ação C) e ordena a instalação de uma entrada OpenFlow para reescrever os pacotes cujos campos de cabeçalho correspondam à regra (uma combinação de uma tupla com 12 campos, neste caso apenas duas: MAC src e dst). Pacotes compatíveis com a descrição da regra recebem o iBF reescrito nos campos MAC sendo encaminhados na rede através das camadas de Aggr e Core tendo como base uma rota na origem (Ação F). Quando o pacote com o iBF chega ao ToR destino, ele verifica a correspondência quanto a entrada de fluxo instalada na Etapa 0 (Ação A) e entrega ao servidor final depois da reescrita dos cabeçalhos MAC (Ação D).

Na Etapa 4, o servidor de destino responde com um TCP SYN ACK o qual, não possuindo uma entrada na tabela de fluxos, é entregue ao controlador (Ação C). Depois da seleção do iBF e da instalação da entrada de fluxo (Ação C), o TCP SYN ACK é encaminhado com base no iBF (Ação F). Após a recepção do pacote no servidor de origem, o *handshake* (aperto de mão) pode ser concluído (Etapa 5) e os dois servidores podem trocar dados na rede sem a intervenção do RM enquanto as entradas se mantiverem ativas.

4.1.5 Ambiente de testes

Dois ambientes de testes foram utilizados para validação da arquitetura SiBF proposta. Ambos utilizam virtualização, mas a principal diferença entre eles é a quantidade de nós possíveis na simulação e a quantidade de máquinas físicas envolvidas.

O primeiro ambiente é composto por 4 máquinas físicas em uma LAN Ethernet, cada máquina hospeda (i) dois controladores NOX (com o RM, TS e DS) e (ii) um nó do banco de dados Keyspace, além das (iii) máquinas virtuais (VM). Os *switches* e servidores são instâncias de VMs, sendo 5 de *switches* OpenFlow e 4 nós finais Debian 4.0 (servidores), configurando um total de 9 VMs em cada máquina física. A Figura 4.7 mostra o primeiro ambiente de testes, onde as linhas contínuas representam ligações diretas entre as máquinas virtuais e as linhas tracejadas representam as conexões entre as máquinas virtuais em diferentes máquinas físicas.

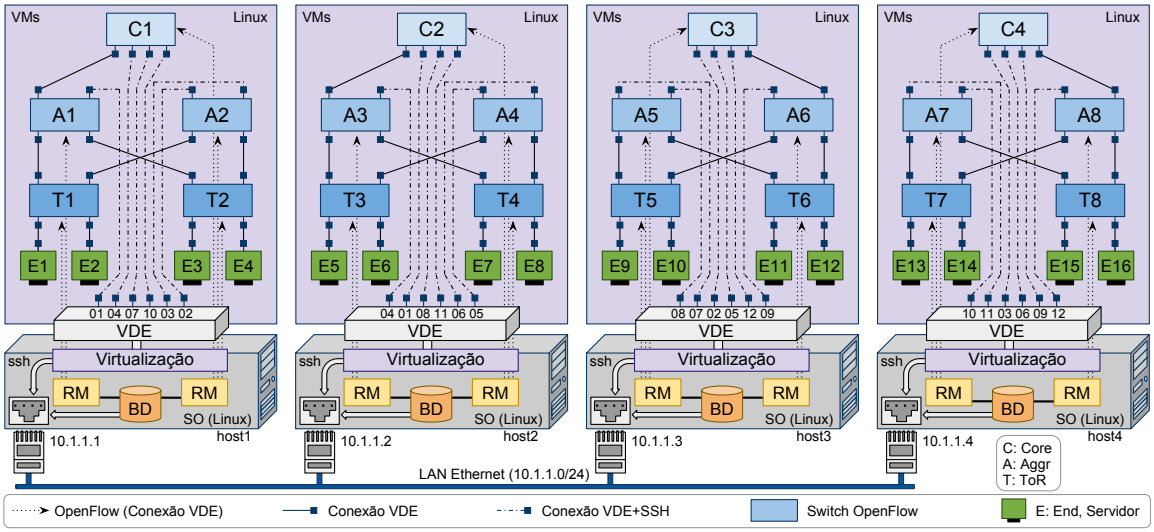


Fig. 4.7: Ambiente de teste (OpenFlowVMS).

A topologia em cada máquina física é configurada com o *OpenFlow Virtual Machine Simulation* (OpenFlowVMS) [33], o qual dispõe de um conjunto útil de *scripts* para automatizar a criação de máquinas virtuais em rede utilizando o QEMU. *Scripts* adicionais foram desenvolvidos para distribuir o ambiente em diferentes máquinas físicas usando *switches* virtuais desprovidos de inteligência e baseados no *Virtual Distributed Ethernet* (VDE) [34] e conexões *Secure Shell* (SSH). O VDE é um *software* que emula interfaces Ethernet, como comutadores e enlaces de comunicação, muito utilizado no ensino, administração e pesquisas em rede de computadores. O *software* não se limita a ambientes virtuais e pode ser implementado em uma ou várias máquinas distribuídas em uma rede local ou até mesmo na Internet. O SSH disponibiliza ao VDE um canal seguro permitindo a comunicação entre máquinas virtuais em diferentes máquinas físicas. As conexões entre máquinas físicas são con-

figuradas com relação de confiança[3] , utilizando chaves assimétricas. Esse mecanismo é necessário para evitar a solicitação de senhas toda vez que uma conexão SSH entre dois VDEs é realizada. O conjunto de *scripts* desenvolvido neste trabalho também permite definir, rapidamente, uma topologia e automatizar a iniciação dos nós virtuais e dos *switches* OpenFlow, incluindo a configuração de IP nos servidores, a criação de *datapath* nos *switches* OpenFlow, a iniciação do módulo do *OpenFlow Protocol* e a conexão ao controlador NOX.

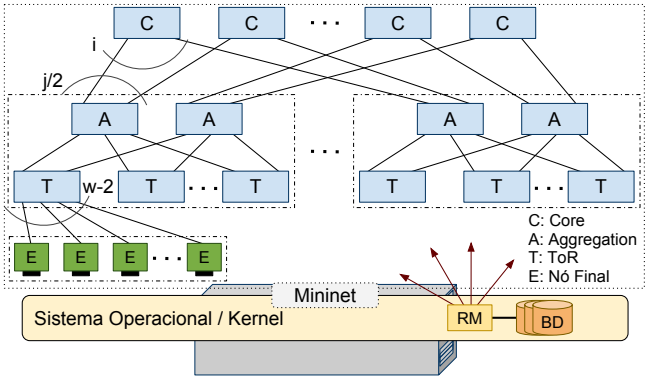


Fig. 4.8: Ambiente de teste (Mininet).

O segundo ambiente de teste foi baseado no Mininet [35]. A grande diferença entre os ambientes virtuais é a possibilidade de criação de um número mais elevado de VMs no Mininet, sem a necessidade de mais de uma máquina física, embora isto seja viável também no Mininet. O objetivo do Mininet é fornecer uma plataforma de virtualização de redes simples e escalável. Isto é possível pois o Mininet trabalha no kernel do sistema operacional criando processos e não máquinas virtuais completas para servidores e *switches* OpenFlow, oferecendo uma medida superior de desempenho comparado com a virtualização tradicional propriamente dita. Dessa forma, o Mininet permite criar facilmente várias topologias de redes com um número mais significativo de equipamentos de redes e nós finais por máquina física. A Figura 4.8 mostra o ambiente de teste do Mininet rodando em um simples nó físico com 8 $Core$, 16 $Aggr$, 32 ToR e 192 nós finais (servidores), totalizando 248 nós na rede, número inviável quando se utiliza a tecnologia OpenFlowVMS (com QEMU). Vale ressaltar que a limitação do número total de *switches* (56) deve-se ao fato da versão atual (alpha) do Mininet não suportar múltiplos controladores OpenFlow. Essa limitação inviabiliza a distribuição do controle como defendido na proposta e exemplificado na Figura 4.7. Apesar dessa limitação, o ambiente do Mininet mostra quão transparente é para a arquitetura o fato do controle estar ou não fisicamente distribuído, onde cada controlador atua como se fosse único na rede. Para o ambiente de teste no Mininet foi criado um *script* que configura automaticamente o número de VMs na topologia *fat tree* através dos parâmetros i, j e w.

[3] A relação de confiança permite que um usuário se conecte ao servidor SSH sem a necessidade de senha.

4.2 Validação e Resultados

Nesta seção, serão discutidos os pontos referentes à validação da arquitetura e dos resultados obtidos a partir do primeiro ambiente de teste (OpenFlowVMS). Depois de validar a implementação do protótipo através da verificação da conectividade total entre o conjunto de servidores (16 VMs), a próxima questão é avaliar o encaminhamento baseado no iBF em termos dos seguintes itens: custo das informações de estado, os potenciais efeitos de falsos positivos e a capacidade de balanceamento de carga. Além disso, outra questão a ser avaliada refere-se à tolerância a falhas. Devido às limitações de um ambiente virtualizado, os aspectos de desempenho não são considerados.

4.2.1 Análise do estado

Seguindo a mesma análise realizada em [36], comparou-se as exigências de estado do SiBF com outras propostas de redes de *data center*: VL2 e PortLand. A configuração topológica da rede é instanciada através de 3 camadas, com ToRs conectados a 20 servidores e a 2 Aggrs, com ligações de 1 e 10 Gbps, respectivamente. As p_1 portas dos *switches* Aggrs são usadas para conexão a $p_1/2$ ToRs e $p_1/2$ Cores com p_2 portas de alta velocidade. Dependendo dos valores de p_1 e p_2, a matriz de interconexão pode ser dimensionada de 3.000 ($p_i = 24$) até 100.000 ($p_i = 144$) nós físicos.

A Tabela 4.1 apresenta os requisitos de escalabilidade para diferentes configurações de *switches*. Em virtude da abordagem rota na origem, SiBF exige um estado mínimo de entradas nos Cores e Aggrs, ou seja, apenas uma entrada por interface. Além disso, a ampliação da rede não afeta o número de entradas de fluxo nos *switches* que é constante e igual ao número de *switches* vizinhos, por exemplo, para um *switch* Core com $p_2 = 24$, o número de entradas na tabela será igual ao número de vizinhos conectados, nesse caso, no máximo 24 entradas. No ToR, a quantidade de entradas de fluxo aumenta com o número simultâneo de fluxos de saída. De acordo com as propostas relacionadas, assumimos uma média de 10 fluxos simultâneos por servidor (5 subindo e 5 descendo), mais o número de entradas constante (uma para cada servidor conectado ao ToR), a fim de executar a reescrita de pacotes e a entrega dos fluxos aos nós finais.

Em comparação, VL2 requer entradas de encaminhamento proporcionais ao número total de *switches* para encaminhar os pacotes ao longo dos dois níveis de encapsulamento IP: $\langle LA_{Core}, LA_{ToR} \rangle$. Por outro lado, o PortLand tem os mesmos requisitos de estado que o SiBF, ou seja, apenas uma entrada de encaminhamento por interface, o suficiente para realizar a transmissão hierárquica sobre os PMACs, mas necessita de um elaborado protocolo de descoberta para determinar a posição exata do *switch* na topologia. Como observação final, uma abordagem de controle direto de rede com base no NOX e OpenFlow pode ser um aliado poderoso e eficiente quando se *repensa* no encaminhamento de pacotes em ambientes gerenciados.

Tab. 4.1: Avaliação das exigências de estado de acordo com o número de entradas nos *switches*.

Nós físicos	2.880			23.040			103.608		
Racks	144			1152			5184		
Switches Aggr.	24 ($p_1 = 24$)			96 ($p_1 = 48$)			144 ($p_1 = 144$)		
Switches Core	12 ($p_2 = 24$)			24 ($p_2 = 96$)			72 ($p_2 = 144$)		
	VL2	PortLand	SiBF	VL2	PortLand	SiBF	VL2	Portland	SiBF
Entradas no ToR	200	120	120	1292	120	120	5420	120	120
Entradas no Aggr	180	24	24	1272	48	48	5400	144	144
Entradas no Core	180	24	24	1272	96	96	5400	144	144

4.2.2 Falsos positivos

Nesta seção avalia-se o desempenho de falso positivo em filtro de Bloom de 96-bits com a presença de apenas 3 elementos identificadores de *switches* para representação de um caminho na rede na topologia do *data center*. É necessário analisar a viabilidade e a eficiência da escolha de projeto para evitar falsos positivos com base no descarte de candidatos iBF propensos a falsos positivos antes da sua utilização. A teoria diz que, dada a baixa taxa de falsos positivos (*false positive rate* (fpr)) de um iBF de 96-bits, há uma abundância de caminhos livres de falsos positivos entre quaisquer ToRs. Para isso, experimentos utilizando uma implementação do iBF no ns-3 (*network simulator*) foram realizados para avaliar na prática o desempenho da fpr. De um conjunto único de 1M de identificadores de 48 bits gerados aleatoriamente, para cada rodada (100.000 no total) foram inseridos 3 desses elementos no iBF de 96 bits. Para cada iBF, foi necessário testar a presença de 432 (= 144 portas $*$ 3 saltos) MACs selecionados aleatoriamente.

Tab. 4.2: Avaliação da taxa de falsos positivos para o iBF de 96 bits.

k	5	6	7	8	9	10	11	13	15	17	19	21
Teor. Eq 3.1 ($\cdot 10^{-6}$)	64.89	25.7	11.68	5.95	3.33	2.03	1.32	0.68	0.42	0.31	0.25	0.23
fpr ($\cdot 10^{-4}$)	2.41	1.81	1.5	1.7	1.83	2.23	3.09	4.92	7.17	11.46	16.09	21.07

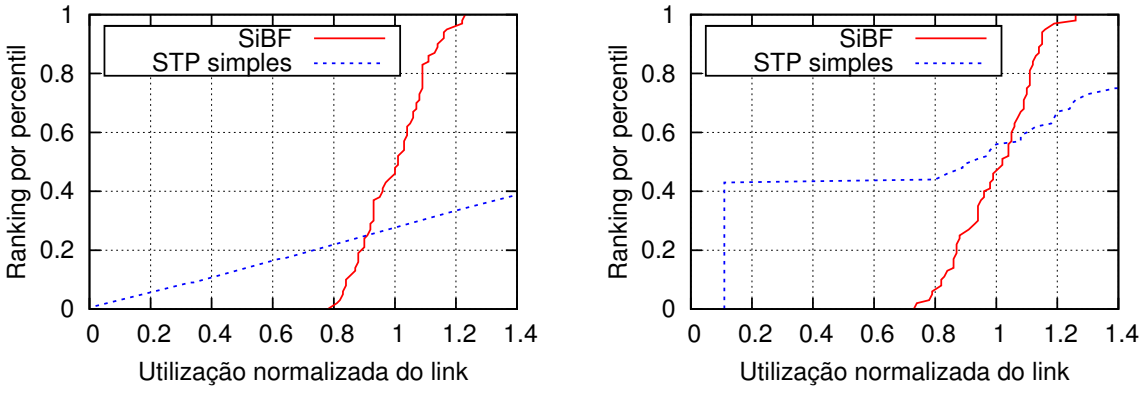
A partir da teoria de filtros de Bloom, existe um número ideal de funções de *hash* ($k_{ideal} = ln2 * m/n$) que minimiza a probabilidade de falsos positivos que, no caso do SiBF (com $m = 96, n = 3$), seria até 22 funções de *hashing*. Na prática, porém, o menor fpr foi obtido para k em torno de 7 ($fpr = 1.5 * 10^{-4}$), conforme Tabela 4.2. A tabela mostra a fpr observada para os experimentos realizados no ns-3 e para os valores teóricos, onde nota-se que os valores teóricos da estimativa da

Eq 3.1 são da ordem de 2 magnitudes menores que os valores práticos. Esse fator é explicado em pesquisas recentes em filtro de Bloom como em [37], onde sugere que a Eq 3.1 não se aplica a todos os valores de m, principalmente para m pequenos, como é o caso da arquitetura SiBF ($m = 96$).

As simulações realizadas no ns-3 sobre topologias de grande escala (mais de 500 *switches* e 10.000 servidores físicos) têm demostrado que essa solução resulta na invalidação, ou seja, na não utilização, de menos de 1% dos múltiplos caminhos disponíveis entre quaisquer dois ToRs. Com isso, em topologias com alto grau de conectividade como a *fat-tree*, em média para *switches* com 48 portas, 94 rotas alternativas (ao invés das 96) são habilitadas para o balanceamento ubíquo do tráfego, sendo uma alternativa mais eficiente que o *Spanning Tree* tradicional nas versões atuais de ECMP (*Equal Cost Multipath Protocol*), que só espalham o tráfego entre 16 caminhos alternativos.

4.2.3 Balanceamento de carga

O balanceamento do tráfego é um requisito importante para o bom funcionamento de um DCN, por isso analisou-se a capacidade de balanceamento de carga utilizando a implementação do VLB com iBF sobre o ambiente de teste do OpenFlowVMS. Dada uma matriz de tráfego TM, o objetivo é avaliar como o tráfego é espalhado entre os enlaces disponíveis. No primeiro momento, comparou-se a utilização do enlace da implementação do SiBF com uma execução simplificada do STP, ambos sobre a mesma topologia. Dois tipos de TMs foram testadas, uma imita um tráfego *todos-para-todos*, comum em aplicações de *data center* (como o MapReduce), e a outra com parâmetros aleatórios. O *Distributed Internet Traffic Generator* (D-ITG) [38] foi utilizado como gerador de tráfego, configurado com fluxos de TCP com duração de 10s e tamanhos de carga exponencialmente distribuídos em torno de 850 bytes. Esses parâmetros são adequados para a maioria dos tráfegos de uma rede de *data center* [39]. A Figura 4.9 mostra a utilização normalizada das ligações (*links*) após a repetição de dez experimentos. Como esperado, na adoção do STP algumas ligações são bastante utilizadas e outras são subutilizadas, enquanto o tráfego SiBF com o VLB apresenta um espalhamento adequado, com a utilização máxima e mínima normalizada de qualquer ligação desviando apenas cerca de 20% do valor ideal, isto é, o valor 1 (um). No caso em que os parâmetros são escolhidos aleatoriamente (Figura 4.9(b)), a conclusão é a mesma, VLB usando iBF consegue um bom aproveitamento dos *links* de forma independente da matriz de tráfego. A distribuição da utilização normalizada das ligações é comparável com os números apresentados na implementação VLB do VL2, com valores mínimos (SiBF - 0.78 vs. 0.46 - VL2) e máximos (SiBF - 1.23 vs 1.2 - VLB) [3, Figura 15]. A divergência nos valores mínimos pode ser explicada pela natureza real do tráfego no VL2 comparado com a matriz de tráfego sintética do SiBF.



(a) Todos para todos.

(b) Aleatório.

Fig. 4.9: Avaliação do comportamento de balanceamento de carga.

4.2.4 Tolerância a falhas e validação experimental

Qualquer sistema de computação está sujeito a falhas e nos *cloud data center*, com milhares de *switches* e servidores, isto é uma situação previsível. Nesta seção, experimentos foram realizados para analisar e validar os pontos da arquitetura SiBF que podem comprometer o funcionamento do *data center*. A Tabela 4.3 mostra uma análise dos testes realizados sobre o comportamento dos fluxos nos ToR, Aggr e Core. Nesta etapa, fluxos foram criados e os elementos em questão na tabela foram derrubados para simular uma falha. Verificou-se o estado dos fluxos em caso de queda de um *switch*, controlador ou nó do Keyspace. Na sequência é apresentada uma discussão sobre os possíveis pontos de falha da arquitetura e alguns resultados da validação prática no ambiente de teste.

Tab. 4.3: Tolerância a falhas quanto a queda de um elemento de rede.

	Fluxo atual			Fluxo novo		
	Controlador	*switch*	Nós BD[1]	Controlador	*switch*	Nó BD[1]
ToR	Mantido	Interrompido	Mantido	Impossível[2]	Impossível	Criado
Aggr	Mantido	Interrompido	Mantido	Criado[3]	Criado[3]	Criado
Core	Mantido	Interrompido	Mantido	Criado[3]	Criado[3]	Criado

[1] BD Keyspace: Podem cair nós na base de dados até que fique um mínimo de 2 nós.

[2] Os pacotes que chegam, são entregues ao nó final.

[3] Criado: Novo fluxo é criado utilizando outra rota.

Rack Manager

Se o *Rack Manager* cair, o atendimento de novos fluxos saindo dos servidores dentro do *rack* fica comprometido pois o RM é responsável por inserir as entradas no ToR que determinam o encaminhamento baseado nos iBFs. Na implementação, o RM é totalmente distribuído e opera de forma independente, o que reduz o escopo afetado para um único *rack*, ou para o conjunto de *switches* sobre os quais o RM atua. Os fluxos correntes não são interrompidos com a queda do RM e para os novos fluxos que têm como destino os servidores agregados ao ToR cujo RM falhou, os pacotes são entregues, mas as respostas não são encaminhadas, já que não existe um controlador para inserir as novas regras.

Após um evento de falha no RM, este pode reiniciar-se em questão de segundos, restabelecer a conexão com o(s) *switch(es)* OpenFlow, recuperar o estado atual da rede acessando os serviços TS e DS e continuar atendendo as requisições de novos fluxo de tráfego. No ambiente de teste (*testbed*) com *round-trip delay time* (RTT) médio de 19 ms (28 ms com percentil 95 após 1000 medições entre combinações de pares de servidores), o tempo médio de atraso adicional do primeiro pacote de um fluxo é de 90 ms (132 ms com percentil 95) devido ao *overhead* de ser redirecionado ao RM até que este retorne a decisão de instalação do fluxo com o mapeamento do iBF. Embora esses tempos não sejam significativos pelas limitações de um *testbed* virtualizado, vale como referência para quantificar como tolerável o *overhead* relativo introduzido pela proposta de controle de rede logicamente centralizado mas fisicamente distribuído.

Topology Service e Directory Service

Os TS e DS também são distribuídos e independentes para garantir o isolamento quando da ocorrência de uma falha. O DS possui um escopo de atuação semelhante ao RM, já que atua nos ToRs. O TS possui um escopo mais amplo, pois atua nos três níveis da topologia *fat-tree*. Por isso, o fator tempo em que o TS fica fora de operação deve ser levado em consideração. Nas falhas transitórias, nada ocorrerá com os fluxos novos e correntes, mas quando as ligações entre switches, mantidas pelo *Tree Discovery* (Seção 4.1.2), sofrerem *timeout*, as entradas dos filtros de Bloom dos vizinhos (Aggr e Core) serão removidas. Na implementação atual, o *timeout* é disparado após o *switch* não responder a três eventos consecutivos de encaminhamento de mensagens LLDP. Atualmente, os pacotes LLDPs são enviados em intervalos de 10 ms. Na implementação atual, todos os componentes estão agregados ao *Rack Manager*. Porém, para uma implementação em maior escala, além de distribuir o RM, os componentes também podem ser desagregados e distribuídos, onde o DS e o RMC atuariam apenas nos ToRs e o TS e *Tree Discovery* atuariam nos três níveis, podendo ter seus próprios gerenciadores.

Base de dados NoSQL

O sistema de base de dados distribuído apresenta um menor impacto, do ponto de vista da ocorrência de falhas, relativamente ao funcionamento da infraestrutura. Deve ainda ser ressaltado que o próprio sistema garante confiabilidade das informações. A maior influência no caso da ocorrência de falhas refere-se à questão do acesso à base de dados, que pode ser feita de forma segura (*safe*) ou "suja" (*dirty*). Métodos *dirty* não oferecem garantias de consistência ao retornar, ou gravar, os valores, mas são mais rápidos. Esses métodos são disponibilizados pelos nós não-mestres (*no-master*) do Keyspace. Os métodos *safe* são disponibilizados apenas pelos nós mestre (*master*), que são mais confiáveis e possuem menos falhas, já que para tornar-se um mestre, o nó precisa apresentar uma maior consistência. Quando um nó mestre cai, um nó não-mestre assume o seu lugar. Dessa forma, as falhas na base de dados são transparentes para o TS e DS.

Switches OpenFlow

Quando um *switch* falhar, seja por um intervalo transitório ou longo, é inevitável que os fluxos correntes passando por ele sejam afetados. Com todos os caminhos habilitados e o uso do VLB para balanceamento de carga, diminuem as chances de *hotspots* ("ponto quente") e o impacto da queda de qualquer *switch* intermediário, já que a carga está balanceada entre todos os *switches* disponíveis, sendo que nenhum deles é crítico para a operação da rede.

Com o encaminhamento baseado nos iBFs inserido nos ToRs conforme as regras de fluxo do OpenFlow, incorporou-se um mecanismo de recuperação de falhas dos *switches* intermediários (Aggr e Core) semelhante ao IP *fast re-route* do *Multi Protocol Label Switching* (MPLS) [40]. A ideia é, com a instalação de cada novo fluxo, também instalar um segundo com prioridade menor, com um iBF alternativo que descreva uma outra rota aleatória, mas paralela, ao primeiro iBF randomicamente selecionado. Com isso, quando ocorrer um evento de queda de *switch* intermediário que afete o caminho principal, o RM deve eliminar no ToR as entradas de prioridade alta afetadas pela falha. Dessa forma, as rotas alternativas começariam a funcionar com o tráfego redirecionado de forma transparente por uma combinação de *switches* Aggr e Core. Esse mecanismo também pode ser explorado no caso da deteção de eventos de congestionamento na rota atual e não necessariamente na falha total de um *switch* intermediário.

Se a conectividade com controlador OpenFlow cair, após um determinado intervalo de tempo os *switches* podem ser configurados para descartar todos os pacotes ou entrar em modo *learning* (por defeito). Esse comportamento padrão prejudica o protocolo de descoberta e não é recomendado na arquitetura proposta pois os pacotes ARP inundariam a malha de *switches* e desabilitariam o protocolo de descoberta, ocasionando *loop* de pacotes LLDP. Outra alternativa seria utilizar o estado de

emergência, onde regras são instaladas e utilizadas apenas em caso de falha do controlador. Uma solução ainda não especificada na versão atual do OpenFlow, consiste na utilização de controladores *slaves* (escravos). Essa e outras questões ainda estão em aberto e encontram-se em discussão na comunidade OpenFlow.

4.3 Resumo do Capítulo

Este capítulo detalhou os métodos adotados na implementação e os tipos de análises realizadas para validação da proposta. Um fator que merece destaque e que ainda não foi mencionado na implementação diz respeito a adoção de tecnologias abertas, ou seja, com código livre disponível para a comunidade, como o protocolo OpenFlow e o controlador NOX. Essa característica fornece dinamismo no desenvolvimento e oferece suporte através de listas de discussão.

O ambiente de teste, seja no OpenFlowVMS ou Mininet, agrega várias tecnologias e oferece um *testbed* eficiente e prover a arquitetura desejada. Por fim, as análises realizadas nesse capítulo demostram que o encaminhamento baseado em filtro de Bloom nos pacotes e implementados seguindo os princípios de projeto é viável, possível e atende aos requisitos de uma rede de *data center*.

Capítulo 5

Considerações Finais

5.1 Conclusões

No cenário atual de serviços em nuvem, as redes de *data center* recebem atenção especial quanto ao provimento de uma infraestrutura capaz de processar serviços sob demanda. Aliado a isso e motivado pelo objetivo de um gerenciamento eficiente da rede através de otimização das funcionalidades, customização das aplicações e rápido processo de inovação, além de oferecer serviços a baixo custo, surge com grande força uma tendência em redes baseada em equipamentos com APIs abertas e interface de controle padronizada. O OpenFlow e o controlador NOX são as tecnologias com maior evidência quando se fala em repensar a rede em relação aos planos de decisão e dados. Por isso, várias propostas estão sendo apresentadas para contornar os desafios que essa nova tendência oferece, como a elaboração de soluções escaláveis e resilientes a falhas, aspectos importantes para serviços críticos de redes como é o caso, por exemplo, do encaminhamento de pacotes.

Nesse contexto, esta dissertação apresenta a proposta de uma nova arquitetura de redes de *data center* que oferece um encaminhamento de pacotes baseado na codificação de rotas na origem por meio de filtro de Bloom nos pacotes, controle direto com visão global da rede e apresenta uma solução para desmistificar o argumento que as redes baseadas no modelo centralizado das redes com OpenFlow possuem um ponto único de falhas, através da introdução de dois serviços escaláveis e tolerantes a falhas. Podemos ainda acrescentar que os resultados obtidos no caso dos serviços de encaminhamento de pacotes e base de dados distribuída podem tornar-se serviços internos ao *data center* seguindo, desta forma, as melhores práticas da programação em nuvem.

5.2 Trabalhos Futuros

Como possíveis trabalhos futuros, pode-se apontar:

Aperfeiçoar a tolerância a falhas: A arquitetura resiliente a falhas necessita de um melhoramento referente a algumas questões de implementação como o tratamento eficiente dos *timeouts* das ligações entre os *switches* vizinhos, quando ocorrem múltiplas falhas. Além de realizar testes para avaliar a eficiência da constante de tempo que determina o grau de atualização dos identificadores dos *switches*, o que influencia na quantidade de atualizações da base de dados. Além disso, pode-se desenvolver e explorar as opções de reencaminhamento dinâmico de fluxos para caminhos alternativos mediante a troca dos iBFs na origem após as falhas nos *switches* ou eventos de congestionamento.

Engenharia de tráfego e balanceamento de carga aleatória: O encaminhamento pode ser melhor implementado levando em conta a engenharia de tráfego, onde é conveniente explorar a eficiência do serviço de encaminhamento de pacotes (alternando com o VLB) na infraestrutura por meio da detecção de fluxos pesados (elefantes), dos eventos de congestionamento (engenharia de tráfego), do uso de técnicas de *multi-path* e *multicast* as quais, em função dos requisitos das aplicações, podem levar a um melhor desempenho. Além disso, técnicas podem ser aplicadas com a engenharia de tráfego para reduzir o consumo de energia (*Green Network*), o que não inviabilizaria o balanceamento de carga, visto que ao desligar equipamentos de redes, a topologia se altera e o protocolo atua para mantê-la atualizada.

***Deletable Bloom filter* (DlBF):** Os possíveis falsos positivos ocasionados com a utilização de Filtros de Bloom podem ser reduzidos adotando este novo membro da família *Bloom Filter*. O pacote ao passar por um *switch* onde o encaminhamento utilizando filtros de Bloom nos pacotes ocorre, a identificação do próximo *switch*, após a tomada de decisão, pode ser deletada do pacote com alta probabilidade diminuindo, dessa forma, o número de bits configurados com o valor 1 (um) e, consequentemente, os possíveis falsos positivos. Além de contornar os falsos positivos, os DlBFs podem ser utilizados para direcionar pacotes a serviços de *middlebox* (com IDs incluídos no iBF) e serem removidos, uma vez que o pacote do fluxo tenha passado por uma instância, independentemente da localização do *middlebox*.

Open vSwitch: Com o advento de *switches* virtuais, abre-se uma gama de possibilidades para implementação de técnicas atualmente inviáveis no *hardware* de *switches* como, por exemplo, a realização do mapeamento diretamente no software de virtualização do *switch* sem a necessidade de atuar diretamente no ToR.

Migração de máquinas virtuais: Para justificar alguns requisitos de *data center* e princípios adotados na arquitetura SiBF é necessária a implementação de um suporte eficiente à migração de máquinas virtuais. Essa característica, apesar de não ter sido implementada, possui suporte nativo, já que a arquitetura não apresenta nenhuma agregação hierárquica quanto aos identificadores de nós finais (virtuais ou físicos).

Extensões OpenFlow: Outra questão importante é incorporar as futuras modificações realizadas pela especificação do padrão OpenFlow, como realizar a verificação bit a bit nos campos dos pacotes, característica que viabilizaria o encaminhamento utilizando filtro de Bloom nos pacotes sem a necessidade de alterar o código original do protocolo OpenFlow. Com essa e outras extensões, abre-se oportunidades para agregar outros serviços à arquitetura, como por exemplo o suporte de *anycast* na comunicação com múltiplos controladores (tal como modelo mestre/escravo) ou a exposição pelo OpenFlow das tabelas L2/L3.

Datacenter distribuído geograficamente: Por fim, mas não esgotando-se o leque de oportunidades, a arquitetura SiBF pode ser agregada a outras tecnologias como *Field-programmable Gate Array* (FPGA) e Eucalyptus, estendida para compor uma nova arquitetura em dois níveis: (1) equipamentos de redes (*Rack Manager*, *switch*) para oferecer conexão interna na rede de *datacenter* (*intra-cloud*) e (2) equipamentos (*gateway*) para conexões entre *datacenter* (*inter-cloud*). Ou seja, a próxima geração de *data center* está caminhando para a interconexão dos atuais *data centers*.

Referências Bibliográficas

[1] Fábio Luciano Verdi, Christian Esteve Rothenberg, Rafael Pasquini, and Mauricio Magalhães. Novas arquiteturas de data center para cloud computing. In *SBRC 2010 - Minicursos*, may 2010.

[2] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. BCube: a high performance, server-centric network architecture for modular data centers. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 63–74, New York, NY, USA, 2009. ACM.

[3] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: a scalable and flexible data center network. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 51–62, New York, NY, USA, 2009. ACM.

[4] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. PortLand: a scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 39–50, New York, NY, USA, 2009. ACM.

[5] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009.

[6] Jeffrey Dean and Sanjay Ghemawat. MapReduce: a flexible data processing tool. *Commun. ACM*, 53(1):72–77, 2010.

[7] Feng Wang, Jie Qiu, Jie Yang, Bo Dong, Xinhui Li, and Ying Li. Hadoop high availability through metadata replication. In *CloudDB '09: Proceeding of the first international workshop on Cloud data management*, pages 37–44, New York, NY, USA, 2009. ACM.

[8] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys '07: Proceedings of the 2nd*

ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, pages 59–72, New York, NY, USA, 2007. ACM.

[9] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, 2008.

[10] Kate Greene. Software-Defined Networking. *MIT technology review*, 112(2):54, 2009.

[11] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. NOX: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, 2008.

[12] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: A distributed control platform for large-scale production networks. In *Proc. of the 9th Symposium on Operating Systems Design and Implementation*, Oct 2010.

[13] Luiz A. Barroso and Urs Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, San Rafael, CA, USA, 2009.

[14] Christian Esteve Rothenberg, Carlos Alberto Bráz Macapuna, Fabio Verdi, Mauricio Magalhães, and Andras Zahemszky. Data center networking with in-packet Bloom filters. In *SBRC 2010*, may 2010.

[15] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, 2007.

[16] Christian Esteve Rothenberg. *Compact forwarding: A probabilistic approach to packet forwarding in content-oriented networks.* PhD thesis, University of Campinas (UNICAMP), December 2010.

[17] The GENI Project Office. GENI: Global Environment for Network Innovations. Página na internet, BBN Technologies, Março 2010. `http://www.geni.net`.

[18] Larry Peterson and Timothy Roscoe. The design principles of planetlab. *SIGOPS Oper. Syst. Rev.*, 40(1):11–16, 2006.

[19] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. Extending networking into the virtualization layer. In *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[20] Albert Greenberg, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Towards a next generation data center architecture: scalability and commoditization. In *PRESTO '08: Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, pages 57–62, New York, NY, USA, 2008. ACM.

[21] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A clean slate 4D approach to network control and management. *SIGCOMM Comput. Commun. Rev.*, 35(5):41–54, 2005.

[22] Raffaele Bolla, Roberto Bruschi, Franco Davoli, and Andrea Ranieri. Energy-aware performance optimization for next-generation green network equipment. In *Proceedings of the 2nd ACM SIGCOMM workshop on Programmable routers for extensible services of tomorrow*, PRESTO '09, pages 49–54, New York, NY, USA, 2009. ACM.

[23] Dilip A. Joseph, Arsalan Tavakoli, and Ion Stoica. A policy-aware switching layer for data centers. *SIGCOMM Comput. Commun. Rev.*, 38(4):51–62, 2008.

[24] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, 2008.

[25] Xin Yuan, Wickus Nienaber, Zhenhai Duan, and Rami Melhem. Oblivious routing for fat-tree based system area networks with uncertain traffic demands. In *SIGMETRICS '07: Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 337–348, New York, NY, USA, 2007. ACM.

[26] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13:422–426, July 1970.

[27] Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel Smid, and Yihui Tang. On the false-positive rate of Bloom filters. *Information Processing Letters*, 108(4):210 – 213, 2008.

[28] Petri Jokela, András Zahemszky, Christian Esteve Rothenberg, Somaya Arianfar, and Pekka Nikander. LIPSIN: line speed publish/subscribe inter-networking. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 195–206, New York, NY, USA, 2009. ACM.

[29] NoSQL. NoSQL: Your Ultimate Guide to the Non - Relational Universe! http://nosql-databases.org/.

[30] Marton Trencseni and Attila Gazso. Keyspace: A Consistently Replicated, Highly-Available Key-Value Store. Whitepaper. http://scalien.com/whitepapers.

[31] Matthew Caesar, Tyson Condie, Jayanthkumar Kannan, Karthik Lakshminarayanan, and Ion Stoica. ROFL: routing on flat labels. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 363–374, New York, NY, USA, 2006. ACM.

[32] Andrei Gurtov, Dmitry Korzun, Andrey Lukyanenko, and Pekka Nikander. Hi3: An efficient and secure networking architecture for mobile hosts. *Comput. Commun.*, 31(10):2457–2467, 2008.

[33] Ben Pfaff and Martin Casado. OpenFlowVMS: OpenFlow Virtual Machine Simulation. Página na internet, OpenFlow Consortium, Fevereiro 2010. `http://www.openflowswitch.org`.

[34] Michael Goldweber and Renzo Davoli. VDE: an emulation environment for supporting computer networking courses. In *ITiCSE '08: Proceedings of the 13th annual conference on Innovation and technology in computer science education*, pages 138–142, New York, USA, 2008. ACM.

[35] Martin Casado, David Erickson, Igor Anatolyevich Ganichev, Rean Griffith, Brandon Heller, Nick Mckeown, Daekyeong Moon, Teemu Koponen, Scott Shenker, and Kyriakos Zarifis. Ripcord: A modular platform for data center networking. Technical Report UCB/EECS-2010-93, EECS Department, University of California, Berkeley, Jun 2010.

[36] Arsalan Tavakoli, Martin Casado, Teemu Koponen, and Scott Shenker. Applying NOX to the datacenter. In *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[37] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang. On the false-positive rate of Bloom filters. *Information Processing Letters*, 108(4):210–213, 2008.

[38] Avallone, Stefano end Botta, Alessio end Dainotti, Alberto end Donato, Walter de and Pescapé, Antonio. D-ITG V. 2.6.1d Manual. `http://www.grid.unina.it/software/ITG`.

[39] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40:92–99, January 2010.

[40] M. Shand and S. Bryant. IP Fast Reroute Framework. RFC 5714 (Informational), January 2010.