Controle distribuído em Redes de Data Center baseado em Gerenciadores de Rack e Serviços de Diretório/Topologia

Carlos A. B. Macapuna, Christian Esteve Rothenberg, Maurício F. Magalhães

[1]Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)
Caixa Postal 6101 – 13083-970 – Campinas – SP – Brasil

{macapuna, chesteve, mauricio}@dca.fee.unicamp.br

Abstract. *In this paper, we present and evaluate the design and implementation of two distributed, 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 promises scalability, performance and fault-tolerance. We show 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, we contribute to demystifying the argument that the centralized controller model of OpenFlow networks is prone to a single point of failure and show that direct network controllers can be physically distributed, yielding thereby a sweet intermediate approach to networking between fully distributed and centralized.*

Resumo. *Neste trabalho, apresentamos e avaliamos a concepção e implementação de dois serviços distribuídos e tolerantes 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 promete escalabilidade, desempenho e tolerância a falhas. O artigo mostra 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, o armazenamento distribuído do par <chave,valor>. Além disso, o artigo contrapõe-se ao argumento de que o modelo de comando centralizado de redes (OpenFlow) está vinculado a um único ponto de falha. Isto é obtido através da proposta de uma arquitetura de controle fisicamente distribuída, mas baseada em uma visão centralizada da rede gerando, desta forma, uma abordagem de controle de rede intermediária, entre totalmente distribuída e centralizada.*

1. Introdução

Como as grades computacionais, a nuvem é um modelo de *utility computing*. Este 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), estas não seriam possíveis sem as tarefas de análise, processamento e armazenamento de dados em grande escala (por exemplo, MapReduce [Dean and Ghemawat 2010], Hadoop [Wang et al. 2009]). 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. Estes e outros requisitos da infraestrutura de TI impostos pelo modelo de *cloud computing* têm motivado a pesquisa de novos projetos e arquiteturas de rede adequadas para os *cloud data centers* de próxima geração [Verdi et al. 2010].

Com o surgimento de switches programáveis baseados na tecnologia OpenFlow [McKeown et al. 2008], emerge um promissor conceito de redes controladas via *software* ou, em inglês, *software-defined networking* [Greene 2009]. O protocolo OpenFlow especifica um caminho padrão para controle das tomadas de decisões no tratamento de pacote (por exemplo, encaminhar na porta x, descartar, enviar para o controlador, etc.) através de uma inteligência (*software*) implementada em controladores logicamente centralizados, mantendo os fornecedores de *hardware* responsáveis apenas pelo desenvolvimento do dispositivo (com suporte à tabela de fluxos especificada pelo OpenFlow). Neste 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, 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* [Barroso and Hölzle 2009], 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 Google ou Amazon, analisem com grande interesse a tecnologia OpenFlow. A grande razão para este interesse reside nos níveis de controle e flexibilidade que eles precisam na provisão dos serviços na nuvem a um custo inferior mas com o mesmo desempenho dos switches *comoditizados* atualmente em operação.

Motivados por esse cenário de desafios e oportunidades, projetamos e implementamos em trabalho anterior [Rothenberg et al. 2010] uma arquitetura de rede de centro de dados (DCN - *data center networks*) que oferece um novo serviço de encaminhamento baseado na codificação de rotas na origem em um filtro de Bloom nos pacotes (iBF - *in-packet Bloom filter*), no caso, adicionado aos campos de endereço do quadro MAC/Ethernet. Neste trabalho, estendemos a proposta para um ambiente de implementação totalmente distribuído. Nesta nova arquitetura são introduzidos dois serviços, escaláveis e tolerantes a falhas, para manter globalmente apenas as informações de topologia e diretório de servidores. Estas informações são mantidas em um sistema de armazenamento distribuído <chave,valor>. Com o desafio de prover uma arquitetura resiliente, o controle centralizado do gerenciador de *rack* (RM - *Rack Manager*) introduzido na nossa arquitetura denominada SiBF [Rothenberg et al. 2010] passa a ser um "exército" de *Rack Managers* com uma instância executando em cada *rack*. Esta abordagem ofe-

rece escalabilidade, desempenho e tolerância a falhas, aproveitando a disponibilidade e as características de bases de dados não relacionais tipicamente disponíveis em *data centers* 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, pode tornar-se mais um serviço da nuvem privada, provendo 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. O artigo mostra 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.

O restante deste trabalho está estruturado conforme descrito a seguir. A Seção 2 apresenta informações relacionadas à arquitetura proposta, assim como, seus princípios. Na Seção 3 são detalhados os principais pontos correspondentes à implementação do protótipo. Na Seção 4 a proposta é avaliada do ponto de vista da tolerância a falhas. Finalmente, na Seção 6, são apresentadas as conclusões e os trabalhos futuros.

2. Arquitetura distribuída proposta

A proposta fundamenta-se na centralização da inteligência de controle da rede (conforme modelo 4D [Greenberg et al. 2005] - *decision*, *dissemination*, *data*, *discovery*) através da remoção da operação distribuída presente em vários protocolos como, por exemplo, a resolução de endereço realizada pelo protocolo ARP. A tomada de decisão é desviada para um elemento controlador e o plano de dados (encaminhamento) é realizado pelos switches. A seguir, descrevemos a topologia, os princípios de projeto e os principais blocos funcionais da nossa proposta.

2.1. Topologia

A distribuição dos equipamentos de rede em um *cloud data center* é tipicamente organizada de acordo com uma topologia *fat-tree*. A opção por esta topologia deve-se à facilidade de implementação através de switches baratos e largamente disponíveis (*commodities*) e por permitir um melhor uso dos recursos da rede [Al-Fares et al. 2008]. A topologia *fat-tree* mostrada na Figura 1 utiliza três camadas: uma camada inferior formada por switches ToR (*Top of Rack*); uma camada intermediária constituída por switches de agregação (Aggr); e uma camada superior de switches de núcleo (Core). Esta 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 infraestrutura como, por exemplo, o *Rack Manager* e o sistema de armazenamento de dados.

2.2. Princípios de projeto

As redes de *data center*, comparativamente à Internet, possuem uma escala muito menor e modelos de controle e gerência que facilitam a adoção de novos paradigmas. Baseado nestas características, os princípios de projeto são relacionados a seguir:

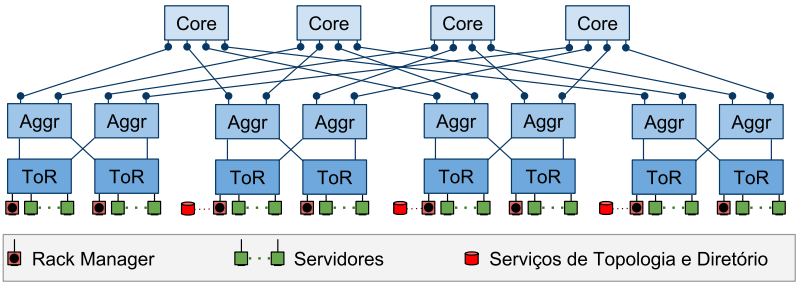

Figure 1. Topologia fat-tree de três camadas.

Separação identificador/localizador: A divisão entre identificador e localizador possui um papel fundamental para viabilizar o compartilhamento dos serviços baseados no endereçamento IP. O IP é utilizado apenas para identificar servidores físicos e as máquinas virtuais (VMs - *virtual machines*) dentro do *data center*. Desta forma, diferentemente da atribuição de endereços IP baseada na hierarquia de provedores adotada na Internet, não são impostas restrições de como os endereços são atribuídos ou utilizados para acesso externo (clientes espalhados na Internet). Esta opção torna o endereço IP não significativo para o encaminhamento de pacotes dentro da infraestrutura do *data center*. Este encaminhamento é realizado no nível da camada 2, modificada para oferecer um serviço de rota na origem de forma transparente aos nós e aplicações legadas.

Rota na origem: De modo a aproveitar o pequeno diâmetro das topologias de redes de *data center*, a abordagem adotada pela arquitetura utiliza o esquema de rota na origem (*strict source routing*). O encaminhamento nas topologias de redes de *data center* em 3 camadas (Figura 1) 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). Esta abordagem permite enviar pacotes por rotas diferentes (balanceamento de carga), além de permitir a seleção da rota menos congestionada (engenharia de tráfego). A especificação da rota na origem é baseada na utilização do filtro de Bloom nos pacotes (iBF - *in-packet Bloom filter*) contendo apenas três elementos identificadores de switches $\langle Core, Aggr_{desce}, ToR_{destino} \rangle$. 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 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 esta informação é implícita ao ToR. Nos próximos saltos, apenas três encaminhamentos são realizados utilizando o iBF, conforme exemplificado na Figura 2.

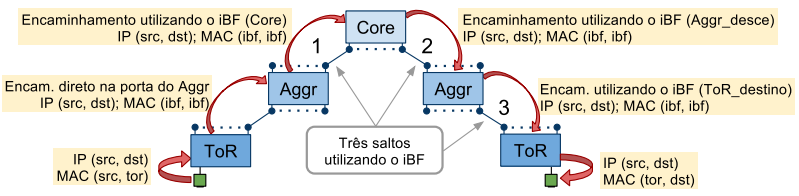

Figure 2. Encaminhamento utilizando iBF.

Controle logicamente centralizado e fisicamente distribuído: Nossa abordagem adota a proposta incluída em 4D [Greenberg et al. 2005] e que sugere a separação da tomada de decisão (controle) do encaminhamento (dados) e introduz um novo elemento de rede, o controlador. Porém, adotamos, o que denominamos de abordagem de rede intermediária, ou seja, um controle logicamente centralizado, onde apenas o estado da topologia e do diretório são mantidos globalmente, mas com controladores distribuídos para atuação sobre um número limitado de switches. Esses controladores são o que chamamos de *Rack Managers* e controlam os switches OpenFlow.

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* [Yuan et al. 2007]). O *Rack Manager* é responsável pela seleção aleatória das rotas compactadas nos iBF na direção do ToR de destino. A estratégia de balanceamento adotada pela arquitetura proposta neste artigo baseia-se no *valiant load balancing* (VLB como usado em [Greenberg et al. 2009]) que roteia de forma aleatória os fluxos na camada de switches intermediários (Aggr e Core).

Tolerância ampla à ocorrência de 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. Considerando um *data center* com 30.000 elementos (servidores e switches), há de se esperar que todo dia tenha, pelo menos, um elemento com mau funcionamento. Neste 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. Este princípio (e o controle logicamente centralizado e fisicamente distribuído) é a principal contribuição deste trabalho em relação ao SiBF e por isso, a avaliação será feita sobre tolerância a falhas, os demais já foram discutidos em [Rothenberg et al. 2010].

2.3. Serviço de encaminhamento 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.[1] A seguir, são apresentados, de forma resumida, o problema e a solução adotada sem falsos positivos.

Filtro de Bloom: Um filtro de Bloom é uma estrutura de dados que identifica se determinado elemento está ou não presente nessa estrutura. É implementado como um vetor de m bits, com n elementos inseridos por k funções de *hash* independentes. Cada elemento, ao ser inserido no filtro, passa por uma função de *hash* cujo 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. Neste caso, não há possibilidade da ocorrência de falsos negativos. Um falso positivo ocorre quando todos os resultados das k funções de *hash* em um elemento que não foi inserido apresentam o valor 1.

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 à cada porta de saída do switch. Cada uma destas entradas corresponde a um filtro de Bloom codificado com a identificação do vizinho detectado. Quando

[1]Mais detalhes do tratamento dos falsos positivos nos iBFs, encontram-se em [Rothenberg et al. 2010].

um pacote com iBF chega ao switch, a máscara de bits é comparada com as entradas na tabela; se uma entrada for encontrada, o pacote é enviado à porta correspondente. Um falso positivo ocasiona o envio do pacote para duas (ou mais) interfaces de saída.

Removendo os falsos positivos: Uma opção para contornar a ocorrência de falsos positivos é a realização do *multicast*. Neste caso, o pacote será encaminhado pelo caminho certo e por mais uma porta e, no próximo switch, com alta probabilidade, será descartado ao carecer de uma entrada válida.[2] Outra opção consiste no encaminhamento *unicast*. Nesta 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, neste caso, que uma nova rota seja definida. No entanto, como o controlador possui a visão global da topologia e existem múltiplos caminhos devido a topologia *fat-tree*, a opção adotada pela arquitetura consiste na eliminação dos caminhos sujeitos a falsos positivos. Desta forma, temos um plano de encaminhamento livre de falsos positivos. As simulações realizadas sobre topologias de grande escala (mais de 500 switches e 10.000 servidores físicos) [Rothenberg et al. 2010] têm demostrado que esta solução resulta na invalidação, ou seja, na não utilização, de menos de 1% dos múltiplos caminhos disponíveis entre quaisquer dois ToRs.

2.4. Serviços de Topologia e Diretório

Assim como outras propostas de novas arquiteturas de *data center* (por exemplo, VL2 [Greenberg et al. 2009] e Portland [Mysore et al. 2009]), 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 ele está ligado. O serviço de topologia é o resultado de um protocolo de descoberta de topologia baseado em uma extensão do LLDP. Ambos os serviços (topologia e diretório) fornecem as informações globais necessárias às funções de controle. O controle do encaminhamento dos fluxos no *data center* tolera um certo grau de inconsistência nas informações fornecidas pelos serviços de topologia e diretório. Esta flexibilidade permite a implementação destes serviços através de um sistema distribuído escalável, confiável e de alto desempenho, do mesmo modo como em muitas outras aplicações na nuvem.

Base de dados não relacionais: A arquitetura proposta no trabalho adota um sistema de base de dados distribuído baseado no par <chave, valor> do tipo NoSQL (*Not only SQL*) [NoSQL]. As implementações nos *cloud data center* utilizando uma estrutura desse tipo apresentam APIs de fácil utilização e atendem os requisitos desejáveis (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 NoSQL 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.

Serviço de Topologia: O conhecimento da topologia é um pré-requisito para o encaminhamento com rota na origem. O *Topology Service* (TS) deve manter atualizado o

[2]Porém, em função da topologia, esta abordagem pode ocasionar *loops*.

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 RMs responsável 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.

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 é eliminada. Portanto, o *Directory Service* (DS) deve manter um mapeamento entre IP e ToR. Esta 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. Para recuperação de falhas do RM, esse mapeamento também é armazenando na base de dados.

3. Implementação

Conforme a ilustração dos componentes na Figura 4, a implementação do encaminhamento dos iBF é baseada em switches OpenFlow, enquanto o *Rack Manager* e os *Topology Service* e *Directory Service* são implementados como aplicações adicionadas ao controlador NOX [Gude et al. 2008]. Já a base de dados estende uma implementação do sistema NoSQL Keyspace [Trencseni and Gazso]. A seguir, descrevemos as principais questões relacionadas à implementação do protótipo e do ambiente de testes.

3.1. OpenFlow

A principal característica do OpenFlow (OF) consiste na definição de uma de uma tabela de fluxos cujas entradas contêm um conjunto de campos do cabeçalho de pacote. Este conjunto é composto por uma tupla formada por 10 elementos 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 a fim 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 OF (v. 0.89rev2 e v 1.0). A Figura 3 mostra a tupla de campos de cabeçalho disponível pelo OF 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.

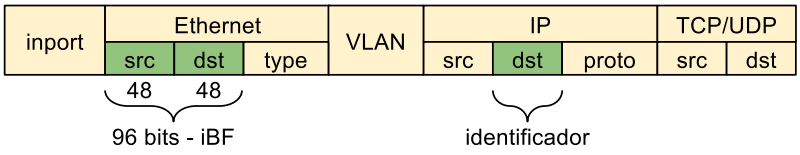

Figure 3. 10 tuplas disponíveis pelo OF e as 3 utilizadas.

3.2. Gerenciador de Rack

O *Rack Manager* (RM) atua como um controlador de switches e a sua implementação é instanciada como um componente que executa no contexto do controlador NOX. 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 OF como `packet-in`, `switch join`, `switch leave`. 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 (ver Figura 4).

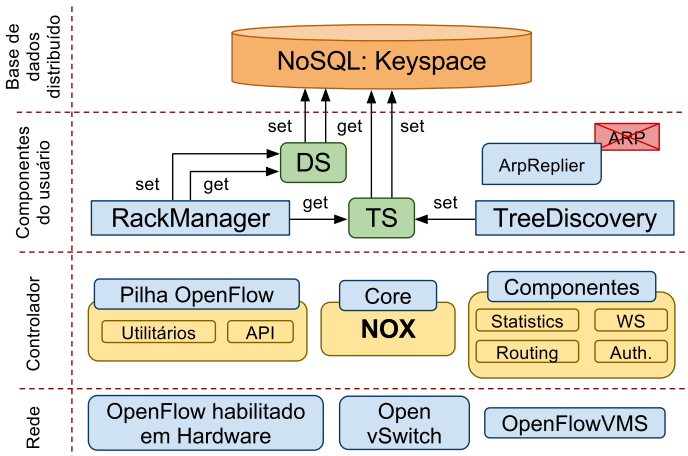

Figure 4. Arquitetura do SiBF.

Rack Manager: É responsável pelo gerenciamento dos novos fluxos que chegam ao switch e a respectiva configuração destes fluxos através dos serviços de diretório e topologia. O *Rack Manager* mantém uma base de dados em *cache* com informações dos servidores descobertos em cada ToR e um mapeamento entre IP, MAC e porta do ToR no qual estes servidores estão anexados. Estas informações são atualizadas no *Directory Service*. O *Rack Manager* interage periodicamente com o *Topology Service* e o *Directory Service* para obter a topologia completa e as informações globais de diretório. Desta forma, o RM constrói uma matriz com os candidatos iBFs livres de falsos positivos para cada caminho entre qualquer combinação de pares de ToRs.

Tree Discovery: Um ponto não trivial é a inferência correta da topologia da árvore e o papel que cada switch desempenha na rede (ou seja, ToR, Core ou Aggr). Para isso, foi desenvolvido um algoritmo que automatiza esta inferência. O *Role Discovery Protocol* utiliza uma extensão do LLDP (*Link Layer Discovery Protocol*) para divulgar a identificação de cada switch. Esta informação é propagada entre os vizinhos, onde cada um descobre qual é o seu papel na topologia (ToR, Aggr ou Core).[3]

Arp Replier: Possui a tarefa de registrar e responder aos pacotes ARP que chegam aos switches de borda, com isso, eliminando a inundação na rede (*flooding*) ocasionada pela propagação de pacotes ARP. Para qualquer requisição de ARP, o *Arp Replier* inclui na resposta, no campo de MAC destino, o endereço MAC associado ao ToR. 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.

[3]Os detalhes do protocolo encontram-se em [Rothenberg et al. 2010].

Topology Service e Directory Service: A implementação do *Directory Service* e *Topology Service* segue uma mesma metodologia. Oferecem métodos de inserção e recuperação 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 mais um prefixo (`set("pre+key", value)`). Isto é necessário pois os dados encontram-se em um único domínio e, neste caso, o prefixo é utilizado para distinguir os tipos de informações (por exemplo, lista de Aggr, mapeamento IP/ToR). Quando uma informação é solicitada, um conjunto de valores é retornado baseado na chave e no prefixo associados. Esta decisão de implementação deve-se às características do esquema de distribuição, já que cada RM armazena na base de dados apenas as suas informações locais, mas necessita obter as informações de toda a topologia. Quando o RM precisa realizar uma consulta, tanto à lista de diretório quanto à de topologia, estas 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 RM. Desta forma, temos uma base de dados logicamente centralizada (mas distribuída fisicamente) e replicada em cada RM.

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 RM 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 RM 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 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*. O quarto contém as identificações internas dos switches utilizadas pelo protocolo de descoberta para criar os *links* entre os vizinhos.

3.3. NoSQL Keyspace

Optou-se pela implementação do sistema distribuído de armazenamento do par <chave,valor> baseado no Keyspace [Trencseni and Gazso]. Além de ser uma implementação em código aberto e em Python, o que facilita a integração com o NOX, o Keyspace oferece consistência, tolerância a falhas e alta disponibilidade. O algoritmo distribuído Paxos é utilizado para manter a consistência dos dados (chave/valores) replicados, onde um quorum de nós verifica a consistência dos dados. Desta forma, pelo menos dois nós devem estar ativos. 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, mesmo, os *Rack Managers*. No ambiente de teste foram 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 [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.

3.4. Ambiente de teste - Testbed

O ambiente de teste utilizado para validação da arquitetura proposta neste artigo é composto por 4 nós físicos em uma LAN Ethernet, cada um hospedando dois controladores

NOX (com o RM, TS e DS) e um nó hospedando o banco de dados Keyspace. Os switches e servidores são VMs, sendo 5 instâncias de switches OF e 4 nós finais Debian 4.0, configurando um total de 9 VMs em cada nó físico. A Figura 5 mostra o *testbed*, onde as linhas sólidas representam ligações diretas entre as máquinas virtuais e as linhas tracejadas representam as conexões entre as máquinas virtuais de diferentes máquinas físicas.

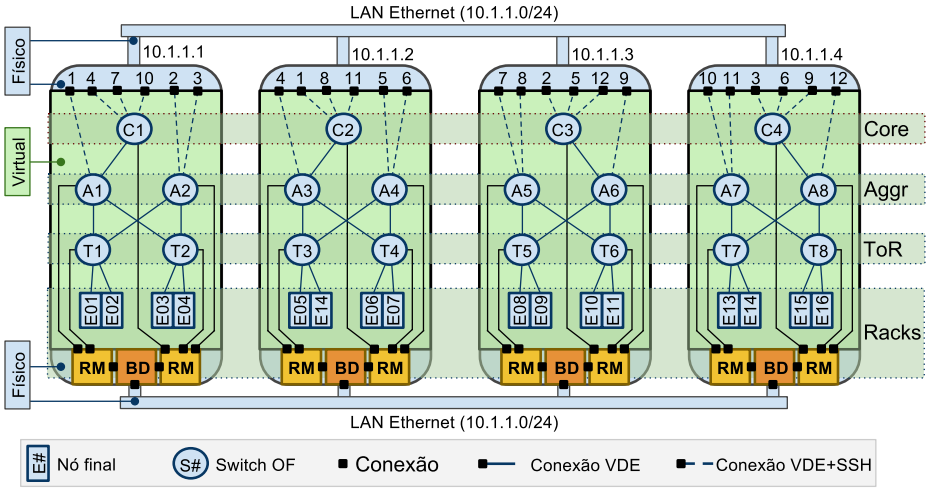

Figure 5. Ambiente de teste.

A topologia em cada máquina física é configurada com o OpenFlowVMS [Pfaff and Casado 2008], 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 VDE (*Virtual Distributed Ethernet* [Goldweber and Davoli 2008]) e conexões SSH (Secure Shell). O VDE é um *software* que emula interfaces Ethernet, como comutadores e linhas de comunicação, muito utilizado no ensino, administração e pesquisas em rede de computadores. Não se limita a ambientes virtuais e pode ser utilizado em uma ou várias máquinas distribuídas em uma rede local ou até mesmo na Internet. Já o SSH disponibiliza ao VDE um canal seguro permitindo a comunicação entre máquinas virtuais em diferentes máquinas físicas. As conexões entre máquinas físicas foram configuradas com relação de confiança, utilizando chaves assimétricas. Este 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 permite definir, rapidamente, uma topologia e automatizar a iniciação dos nós virtuais e do OF switch, incluindo a configuração de IP nos servidores, a criação de *data path* nos switches OF, a iniciação do módulo do *OpenFlow Protocol* e a conexão ao controlador.

4. Tolerância a falhas e validação experimental

Qualquer sistema de computação está sujeito a falhas, nos *cloud data center*, com milhares de switches e servidores, isto é uma situação previsível. Nesta seção analisamos e validamos experimentalmente, aqueles pontos da arquitetura proposta que podem comprometer o funcionamento do *data center*. Assim como em [Rothenberg et al. 2010], neste trabalho também não avaliamos a arquitetura quanto ao desempenho e escalabilidade, pelas limitações do ambiente virtual.

A Tabela 1 mostra uma análise dos testes realizados (avaliação prática) sobre o comportamento dos fluxos nos ToR, Aggr e Core. Verificamos o estado dos fluxos caso um switch, controlador ou nó do Keyspace venha falhar. Na sequência é apresentada uma discussão sobre os possíveis pontos de falha da arquitetura e alguns resultados da validação prática do *testbed*.

Table 1. 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 discutida no artigo, o RM é totalmente distribuído e opera de forma independente. Desta forma, o escopo afetado é reduzido para um único *rack*. 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 *testbed* com RTT (*round-trip delay time*) 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 estes 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 3.2), 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.

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 nas informação. 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. Estes métodos são disponibilizados pelos nós não-mestre (*no-master*) do Keyspace. Os métodos *safes* são disponibilizados apenas pelos mestres, que são mais confiáveis e possuem menos falhas. Quando um *master* cai, um nó não-mestre assume o seu lugar. Desta 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* e o impacto da queda de qualquer switch intermediário, já que a carga está balanceada entre todos os switches disponíveis.

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 MPLS [Shand and Bryant 2010]. 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. Desta forma, 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. Desta forma, as alternativas começariam a funcionar com o tráfego redirecionado de forma transparente por uma combinação de switches Aggr e Core. Este 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). Este comportamento padrão prejudica o protocolo de descoberta e não é recomendado na arquitetura proposta pois os ARPs 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). Esta e outras questões encontram-se em discussão na comunidade OpenFlow.

5. Trabalhos relacionados

O VL2 [Greenberg et al. 2009] é uma proposta de arquitetura de *data center* que oferece uma camada 2 virtual escalável. Implementa um *Directory System* (DS) que provê três funções: consultas, atualizações e mapeamentos. Apoiado nas diferentes exigências de desempenho, O DS do VL2 possui duas camadas de servidores replicados, uma otimizada para leitura e mapeamento (*directory servers*) que deve oferecer baixa latência e alta disponibilidade, e outra otimizada para escrita (*replicated state machine*), que utiliza o algoritmo de consenso Paxos para garantir a consistência das informações. Essa otimiza-

ção, onde há mais consulta do que escrita, é realizada no DS da arquitetura proposta através dos métodos *dirty* (para leitura) e *safe* (para escrita) disponíveis no Keyspace.

O Portland [Mysore et al. 2009] propõe um encaminhamento baseado na posição de pseudo endereço MAC (PMAC) atribuído aos nós finais para oferecer escalabilidade sem alterar o endereçamento Ethernet. Para isso, desenvolveu-se um protocolo de descoberta de posição dos switches na topologia (LDP) e um serviço logicamente centralizado de gerenciamento da infraestrutura de rede (*Fabric Manager* - FM). O FM mantém um estado da topologia da rede e um mapeamento entre o endereço MAC real do nó e um PMAC. Tanto o LDP quanto o FM possuem características semelhantes à arquitetura apresentada neste artigo como, por exemplo, o conhecimento prévio das posições dos switches para o encaminhamento e a manutenção de um estado global da rede.

6. Considerações finais e trabalhos futuros

No cenário tecnológico de *data centers* provendo serviços em nuvem e motivados pelos objetivos de controle (otimização das funcionalidades, customização dos serviços e aplicações, rápido processo de inovação) e baixo custo, aparece com força uma tendência de redes baseadas em equipamentos aderentes a interfaces de controle padronizadas. Porém, até esse cenário se tornar uma realidade operacional, vários desafios terão que ser contornados, entre eles, a provisão de soluções escaláveis e tolerantes a falhas, dois aspectos especialmente críticos para um serviço de rede como é o de encaminhamento de pacotes. Com esse objetivo, o presente artigo estende a proposta de arquitetura de *data centers* baseada na codificação de rotas na origem por meio de filtros de Bloom nos pacotes através da introdução de dois serviços escaláveis e tolerantes a falhas, os serviços de Diretório e Topologia, que se juntam a uma distribuição física dos elementos de controle da rede (*Rack Managers*). A distribuição do estado global da topologia e diretório torna o ambiente altamente escalável e tolerante a falhas, dois fatos corroborados pelas avaliações analíticas e experimentais apresentadas neste artigo. Em trabalhos futuros, algumas questões ligadas à tolerância a falhas deverão ser explorados com maior detalhe como, por exemplo, o tratamento eficiente dos *timeouts* nas ligações entre switches vizinhos quando da ocorrência de múltiplas falhas. Partindo da opção adotada neste trabalho por uma arquitetura de controle intermediária entre as opções totalmente centralizada e totalmente distribuída, os objetivos futuros do presente trabalho têm como foco o suporte eficiente à migração de máquinas virtuais. Esta migração eficiente é um requisito fundamental para o funcionamento otimizado dos futuros *cloud data centers*. Por último, é importante frisar que a arquitetura procurará incorporar as futuras extensões que serão agregadas à especificação do padrão OpenFlow.

Referências

Al-Fares, M., Loukissas, A., and Vahdat, A. (2008). A scalable, commodity data center network architecture. *SIGCOMM CCR*, 38(4):63–74.

Barroso, L. A. and Hölzle, U. (2009). *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, San Rafael, CA, USA.

Dean, J. and Ghemawat, S. (2010). MapReduce: a flexible data processing tool. *Commun. ACM*, 53(1):72–77.

Goldweber, M. and Davoli, R. (2008). VDE: an emulation environment for supporting computer networking courses. In *ITiCSE '08*, pages 138–142, New York, NY, USA.

Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. (2009). VL2: a scalable and flexible data center network. In *SIGCOMM '09*, pages 51–62, New York, NY, USA. ACM.

Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and Zhang, H. (2005). A clean slate 4D approach to network control and management. *SIGCOMM CCR*, 35(5):41–54.

Greene, K. (2009). Software-Defined Networking. *MIT technology review*, 112(2):54.

Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). NOX: towards an operating system for networks. *SIGCOMM CCR*.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *SIGCOMM CCR*, 38(2):69–74.

Mysore, R. N., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V., and Vahdat, A. (2009). PortLand: a scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM '09*, pages 39–50, New York, NY, USA.

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

Pfaff, B. and Casado, M. (2008). OpenFlowVMS: OpenFlow Virtual Machine Simulation. http://www.openflowswitch.org. Acesso em 25/02/2010.

Rothenberg, C. E., Macapuna, C. A. B., Verdi, F., Magalhães, M., and Zahemszky, A. (2010). Data center networking with in-packet Bloom filters. In *SBRC 2010*.

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

Trencseni, M. and Gazso, A. Keyspace: A Consistently Replicated, Highly-Available Key-Value Store. Whitepaper. http://scalien.com/whitepapers.

Verdi, F. L., Rothenberg, C. E., Pasquini, R., and Magalhães, M. (2010). Novas arquiteturas de data center para cloud computing. In *SBRC 2010 - Minicursos*.

Wang, F., Qiu, J., Yang, J., Dong, B., Li, X., and Li, Y. (2009). Hadoop high availability through metadata replication. In *CloudDB '09*, pages 37–44, New York, NY, USA.

Yuan, X., Nienaber, W., Duan, Z., and Melhem, R. (2007). Oblivious routing for fat-tree based system area networks with uncertain traffic demands. In *SIGMETRICS '07*, pages 337–348, New York, NY, USA. ACM.