PathFlow: uma solução SDN para o problema de migração de máquinas virtuais em *Data Centers*

Gilvan de Almeida C. Filho[1], **Sidney C. de Lucena**[1]

[1]Departamento de Informática Aplicada / PPGI
Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
Rio de Janeiro – RJ – Brazil

{gilvan.filho,sidney}@uniriotec.br

Abstract. *Server virtualization in Data Centers has enabled the implementation of several Cloud Computing features. Nowadays virtualization softwares allow online migration of virtual machines (VMs) so that they may adapt themselves to new traffic and performance demands. The VM's IP must be maintained during the migration, which limits the migration to occur within the same network. This work addresses the problem of online migration of virtual machines between different networks by proposing a path switching SDN-based solution, called PathFlow. Obtained results show the feasibility and effectiveness of the solution, with a significant reduction in switching tables size, improving scalability over traditional solutions.*

Resumo. *A virtualização de servidores em Data Centers permitiu a implementação de diversas funcionalidades de computação em nuvem. Os softwares atuais de virtualização permitem a migração on-line de máquinas virtuais (VMs) de maneira que esses se adaptem a novas demandas de tráfego e desempenho. Para tal, o IP da VM deve ser mantido durante a migração, o que limita a migração para que essa ocorra dentro de uma mesma rede. Este trabalho endereça o problema de migração on-line de máquinas virtuais entre redes distintas ao propor uma solução SDN de comutação baseada em caminhos, chamada PathFlow. Os resultados obtidos mostram a exequibilidade e efetividade da solução, assim como significativa redução das tabelas de comutação, trazendo uma maior escalabilidade em relação a soluções tradicionais.*

1. Introdução

Os *Data Centers* vêm ganhando importância nos últimos anos à medida que aumenta a demanda por infraestrutura de armazenamento e hospedagem de serviços acessados em larga escala. Relacionado a isto, o modelo de *Cloud Computing* torna-se fundamental para a disponibilização de serviços sob demanda a partir da alocação dinâmica de recursos computacionais e de rede nos *Data Centers*. Esta alocação dinâmica faz uso intenso de virtualização, onde múltiplas máquinas virtuais (*virtual machines* – VMs) são alocadas em um único servidor físico (*host*) e frequentemente migradas para outros servidores, seja por necessidade de manutenção parcial do *Data Center* ou para acomodar novas demandas de tráfego.

Os softwares atuais de virtualização (hipervisores) permitem a migração *on-line* de VMs, que consiste em mudá-las de *host* sem que as VMs sejam desligadas. Isso é feito

dinamicamente, gravando o estado da memória da VM na origem e incrementalmente transferindo-o para o *host* destino. E para evitar quedas de conexão entre processos clientes e o respectivo serviço, o endereço IP da VM deve ser mantido durante a migração. Para isto, é necessário que a nova localização esteja conectada ao mesmo domínio de *broadcast* da anterior (por exemplo, em outra porta do mesmo *switch*), o que pode demandar ajustes na configuração de VLANs em dispositivos de nível 2. Tais ajustes podem se tornar bastante desafiadores quando a mobilidade ocorra entre redes WAN, mas também entre sub-redes no próprio Data Center.

Para que a migração de máquinas virtuais entre redes distintas ocorra de forma eficiente, grande parte das soluções utiliza técnicas de tunelamento, como VXLAN [Sridhar et al. 2013], NVGRE[Sridharan et al. 2011] e LISP[Farinacci et al. 2013]. A ideia básica consiste em estender o nível 2 (LAN) da rede origem para a rede destino por meio destes túneis, inclusive através de WANs. No entanto, principal desvantagem dessas soluções é que o tráfego destinado a um *host* móvel precisará passar por um ponto de âncora na rede origem, para então ser direcionado ao túnel e assim alcançar o *host* na rede destino, com o tráfego de resposta percorrendo o caminho inverso. Esta ineficiente triangulação do tráfego, além de sobrecarregar o túnel, traz severas restrições de escalabilidade.

O paradigma do SDN (Software Defined Network) propõe uma forma mais ágil de se implantar inovação na rede. O SDN [McKeown 2011] é uma arquitetura emergente onde a funcionalidade de controle da rede (plano de controle) é destacada do *hardware* que realiza a comutação (plano de dados), permitindo a programabilidade da rede. O controle, que era integrado fortemente nos equipamentos individuais de rede, passa a ser realizado por controladores SDN logicamente centralizados, permitindo a abstração da infraestrutura de rede para as aplicações e serviços de rede. Dessa forma, aplicações podem tratar a rede como uma entidade lógica ou virtual.

Diante das limitações impostas pela arquitetura de rede tradicional, este trabalho propõe uma solução SDN de comutação baseada em caminhos para o problema de migração de VMs em *Data Centers*, chamada **PathFlow**. Esta solução endereça o problema de migração de VMs tomando como base recentes trabalhos de virtualização de redes [Chowdhury and Boutaba 2010] que permitem um *switch* ou uma rede inteira serem logicamente independentes dos dispositivos físicos. A arquitetura de nossa proposta enxerga a rede como uma entidade única formada por elementos de encaminhamento (*forwarding elements* - FEs) que são gerenciados por um controle central (*central control* - CC). O PathFlow fundamenta-se no fato de que o controlador SDN possui uma visão holística da rede, conhecendo todos os FE, que no nosso caso são *switches* de rede, e as ligações entre eles, conseguindo dessa forma criar e manter uma topologia e calcular caminhos (***paths***) mínimos. Os resultados obtidos para o PathFlow mostram a exequibilidade e a efetividade da solução, que será detalhada no decorrer deste trabalho.

Este artigo está organizado conforme descrito a seguir. Na Seção 2 serão revistos os principais trabalhos relacionados. A Seção 3 sintetiza a arquitetura SDN e sua implementação através do protocolo OpenFlow. Na Seção 4 será apresentado o PathFlow, sua operação e arquitetura. Na Seção 5 serão detalhados os testes de validação da proposta. A Seção 6 finaliza este trabalho trazendo a conclusão do artigo e os trabalhos futuros.

2. Trabalhos Relacionados

O problema de migração de VMs em *Data Centers* guarda relação com o tema de mobilidade em redes TCP/IP, uma vez que ambos os casos tratam da manutenção do endereço IP a medida que o dispositivo (VM) se move (migra). Uma das primeiras abordagens sobre mobilidade IP foi normatizada pelo IETF como adendo à especificação do IPv4[Perkins 2002] para comunicação destinada a dispositivos móveis, permitindo ao nó móvel alterar seu ponto de interligação com a rede sem, no entanto, alterar seu endereço IP. Dessa forma, as aplicações poderiam manter suas conexões ativas durante a movimentação do nó. Para tal, permiti-se ao nó móvel possuir dois endereços IP: um fixo (*home address*) associado ao endereço da sua rede original (*home network*), e outro temporário baseado no endereço da rede para a qual o nó migrou. O protocolo IP móvel realiza uma associação entre os dois endereços através de agentes que rodam no nó móvel (*home agent* - HA) e na rede (*foreign agent* - FA). Quando o nó móvel está em outra rede que não a sua rede original, HA e FA estabelecem um túnel para comunicação entre o endereço fixo e o temporário. Essa abordagem causa uma ineficiente triangulação do tráfego, pois o tráfego destinado a um dispositivo precisaria ser redirecionado a um ponto de âncora (HA) para depois ser repassado ao destino móvel. Trata-se de uma solução aplicável a uma rede IP móvel cujo tráfego é relativamente baixo, mas não no contexto de um *Data Center* em que servidores lidam com volumes de tráfego muito maiores.

Uma solução desse problema de ancoragem é prevista na especificação do IPv6[Johnson 2004], ao definir um protocolo de rastreamento de nós móveis. Esse protocolo, chamado IPV6 móvel (*mobile IPv6*), permite aos nós móveis migrarem de uma rede para outra sem mudar seu *home address*. Pacotes podem ser roteados ao nó móvel independente de sua localização à rede, através da troca de informações previstas no protocolo. Porém, para que essa proposta seja implementada, todos os nós do *Data Center* precisariam suportar o protocolo IPv6, o que leva a outros problemas, dado que a transição de IPv4 para IPv6 não ocorre em pouco tempo e não se pode exigir que todos os nós de um *Data Center* suportem esse protocolo.

Foram também propostos alguns protocolos de suporte à mobilidade na camada de transporte, como o SCTP [Stewart 2007], o TCP-MH [Matsumoto et al. 2003] e o DCCP [Floyd et al. 2006], e há também o suporte a mobilidade em camadas mais acima, como o Snoeren [Snoeren 2002] e o MSOCKS [Maltz et al. 1998]. Estas soluções possuem a desvantagem de necessitarem de alterações nos *hosts* para permitir a funcionalidade de controle ou execução da migração. Isso tende a ser um problema em um *Data Center*, que necessita suportar uma diversidade de tipos de *hosts*.

A mobilidade também pode ocorrer em camadas abaixo à do IP. Na camada de enlace, a solução da Cisco chamada *Local Area Mobility* (LAM [CISCO 2015]) permite que nós móveis migrem de sua sub-rede origem para outra localidade, dentro da mesma organização, mantendo a conectividade transparente. Embora não dependa de alteração de *hosts*, técnicas como essa possuem pouca eficácia à medida que uma rede aumenta de tamanho e de número de *hosts*, pois para cada nó móvel uma rota específica precisa ser instalada e divulgada, causando problemas de partição das tabelas de roteamento.

No contexto de mobilidade, redes sobrepostas são amplamente utilizadas atualmente para interconexão de redes. Uma rede sobreposta é uma rede virtual que é criada acima de uma rede física. Os nós em uma rede sobreposta são interconectados através de

links virtuais (túneis), que correspondem a caminhos na rede física. Um dispositivo na borda de uma rede (em *hardware* ou *software*) é então dinamicamente programado para gerenciar túneis entre os hipervisores e/ou *switches* da rede. Diversas tecnologias de tunelamento são utilizadas para o estabelecimento e controle destes túneis, onde podem ser destacados o *Stateless Transport Tunneling* (STT) [Davie and Gross 2013], o *Virtualized Layer 2 Networks* (VXLAN) [Sridhar et al. 2013], o *Network Virtualization Using Generic Routing Encapsulation* (NVGRE) [Sridharan et al. 2011], o *Locator/ID Separation Protocol* (LISP) [Farinacci et al. 2013] e o *Generic Network Virtualization Encapsulation* (GENEVE) [Gross et al. 2014]. Embora flexíveis, estas redes sobrepostas possuem limitações, como dito em Anderson et al [Anderson et al. 2005], que pondera que as tecnologias de sobreposição não podem ser consideradas como caminho de implantação para tecnologias disruptivas. Primeiramente, elas são utilizadas para resolver limitações pontuais sem uma visão holística da rede, nem interações entre as sobreposições. Em segundo, muitas das sobreposições estão sendo construídas na camada de aplicação, sendo, portanto, incapazes de suportar radicalmente diferentes arquiteturas.

Propostas baseadas em SDN estão sendo apresentadas endereçando o problema de mobilidade. O VICTOR (*Virtually Clustered Open Router*) [Hao et al. 2009] é uma arquitetura de rede que endereça o problema de migração de VMS entre diversas redes distintas, permitindo a manutenção de seus endereços IP originais. A ideia principal do VICTOR, como mostrado na Figura 1 (baseada em [Hao et al. 2009]), é criar um *cluster* de elementos de encaminhamento (FE) que funcionam como placas (*line cards*) com múltiplas portas virtuais de um único roteador virtualizado. Dessa forma, a agregação de FEs realiza o encaminhamento de dados para o tráfego de uma rede. Os FEs são distribuídos ao longo de várias redes, o que ajuda a suportar a migração de máquinas virtuais através dessas redes. A principal limitação do VICTOR é que ele requer suporte a bases de informação de encaminhamento (*Forwarding Information Base* - FIBs) de grande tamanho, levando a preocupações quanto à escalabilidade da solução [Bari et al. 2013].

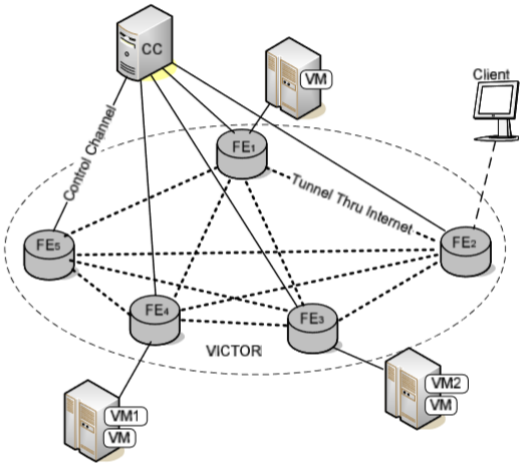


Figura 1. Arquitetura do VICTOR

3. SDN e OpenFlow

Redes definidas por software (*Software-Defined Networking* - SDN) [McKeown 2011, Shenker et al. 2011] é um paradigma emergente que rompe as limitações arquiteturais

das redes atuais. Inicialmente, ele quebra a tradicional integração vertical dos dispositivos de rede, separando o plano de controle do plano de dados e fazendo com que roteadores e *switches* tornem-se simples dispositivos de encaminhamento de tráfego. O controle lógico passa a ser efetuado por um **controlador** centralizado que transfere regras de encaminhamento para os *switches*, simplificando, dessa forma, a aplicação de políticas, a reconfiguração e a evolução da rede[Kim and Feamster 2013].

O Openflow foi proposto para padronizar a comunicação entre controladores e *switches* em uma arquitetura SDN [Lara et al. 2014], definindo também o protocolo dessa comunicação. O Openflow, dessa forma, provê formas de controlar os dispositivos de rede, que no caso são genericamente denominados *switches* Openflow, ou *datapaths*. Através do protocolo OpenFlow, os controladores podem programar as chamadas tabelas de fluxo (*flow tables*) destes *switches*. As regras de encaminhamento de fluxos podem usar parâmetros de várias camadas, flexibilizando também a associação entre os mesmos.

O Openflow é mantido pelo *Open Network Foundation* - ONF [ONF 2015]. A versão 1.0, lançada em dezembro de 2009, é atualmente a mais difundida e implementada. Após a versão 1.0, foram lançadas as versões 1.1, 1.2, 1.3, 1.4, 1.5 e atualmente se encontra na versão 1.6 [Lara et al. 2014]. A tabela de fluxo da versão 1.0 possui 12 campos que são utilizados para a classificação de um fluxo, conforme mostrado na Tabela 1. Cada entrada na tabela de fluxos é associada a uma ou mais ações. As ações mais comuns envolvem encaminhar o pacote para uma ou mais interfaces de saída, enviar o quadro para o controlador, modificar um ou mais campos, ou ainda descartar o quadro. Para cada regra na tabela de fluxo, há contadores que incrementam a cada pacote que casa com a regra, sendo bastante úteis para fins de estatística.

Tabela 1. Campos do pacote utilizados para correspondência com entradas de fluxos

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

4. Proposta de Solução

Não obstante a sua complexidade, os *Data Centers* normalmente têm topologias simples e hierárquicas onde uma pequena quantidade de saltos é suficiente para um quadro atravessar toda a rede. Dessa forma, um controlador SDN consegue calcular facilmente todos os caminhos que interligam os *switches* de um *Data Center*, sendo essa quantidade bem menor que a quantidade de endereços MAC totais que os equipamentos de núcleo precisam manter em sua tabelas. Como há um conhecimento prévio da topologia do *Data Center*, consequentemente há como limitar em projeto a quantidade de caminhos totais.

4.1. PathFlow

O PathFlow se apoia no fato de que o controlador SDN possui uma visão holística da rede, ou seja, conhece todos os FE, incluindo *switches* virtuais, e as ligações entre eles, conseguindo dessa forma criar e manter uma topologia e calcular os menores caminhos de encaminhamento. Além da topologia, o PathFlow mapeia em quais *switches* estão ligados todos os *hosts* da rede e relaciona, para cada *switch* origem, qual o caminho (*path*) que deve ser tomado para se alcançar o *switch* destino em que cada *host* está ligado. O *switch*

origem então insere um ***tag*** identificador do caminho no pacote. A comutação em todos os *switches* intermediários se dá por esse *tag* até alcançar o *switch* destino, que o retira e entrega ao *host*.

O quadro Ethernet ao chegar ao *switch* SDN, seja físico ou virtual, é alterado e é incluído o rótulo do caminho que ele precisa pegar para chegar ao destino, conforme a Figura 2. Dessa forma, cada *switch* origem precisa conhecer apenas o caminho (*path*) para cada um dos *hosts* destinos. Adicionalmente, um *switch* não precisa conhecer todos os caminhos para todos os *hosts* da rede, apenas aqueles para os quais o *host* origem tem interesse de tráfego. Isso reduz o impacto de escalabilidade à medida que há um grande crescimento do número de VMs (*hosts*) no *Data Center*. Nos *switches* intermediários, tipicamente *switches* de agregação ou núcleo que possuem poucos *hosts* diretamente ligados, as tabelas de comutação ficam menores, pois recebem do controlador SDN basicamente as informações de comutação dos *paths*.

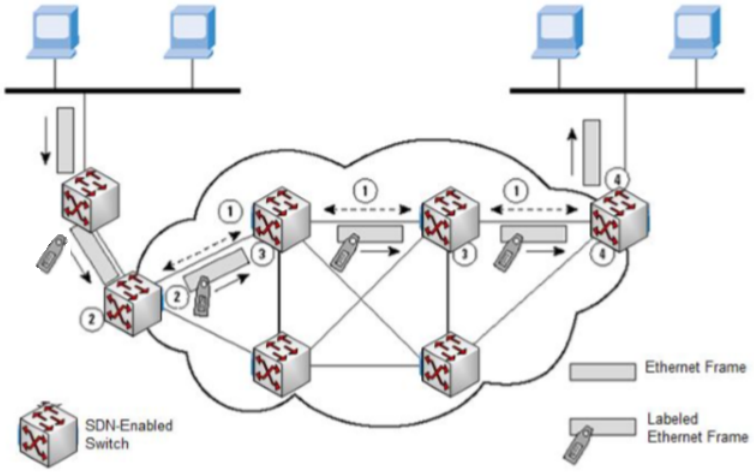


Figura 2. Comutação baseada em caminhos

A ideia de se utilizar comutação baseada em caminhos não é em si uma novidade. Essa técnica é a base do MPLS (*Multi Protocol Label Switching*). A diferença principal é que o MPLS possui uma arquitetura distribuída em que as funções são implementadas no mesmo dispositivo onde também há a função de comutação. Por se estar propondo uma solução SDN, as funções de controle serão centralizadas e implementadas pelo controlador SDN. O equivalente à função LDP (*Label Distribution Protocol*) do MPLS, por exemplo, que é responsável pela distribuição e manutenção dos rótulos de comutação, é gerenciado centralmente pelo controlador SDN.

No caso de um *host* ser movido de um *switch* a outro, por movimentação física ou virtual no caso de migração de uma VM, o controlador SDN notifica os demais *switches* da rede para alterarem o caminho para os quais deve comutar o pacote para alcançar novo *switch* destino. Essa alteração se dá simplesmente alterando o *path* para o destino e pode ser acionada tanto quando o controlador SDN é notificado da nova posição como, adicionalmente, recebendo uma mensagem ou diretiva do hipervisor.

O PathFlow permite, dessa forma, que o *host* seja alcançado em qualquer ponto da infraestrutura da rede gerenciada pelo sistema, independentemente de sua posição. Dife-

rentemente de uma rede tradicional em que as informações das subredes são descentralizadas e estão gravadas nos *switches* da infraestrutura, o PathFlow gerencia as informações de forma centralizada. Essa sistemática permite a livre movimentação do *host* para qualquer parte da rede, independentemente da subrede a que pertença, sem que seja necessária alteração de IP nem que sejam interrompidas as conexões a esse *host*.

4.1.1. Comutação baseada em caminhos

A Figura 3 ilustra a diferença que existe no comportamento de aprendizagem em um *switch* Ethernet tradicional e num *switch* de comutação baseada em caminhos.

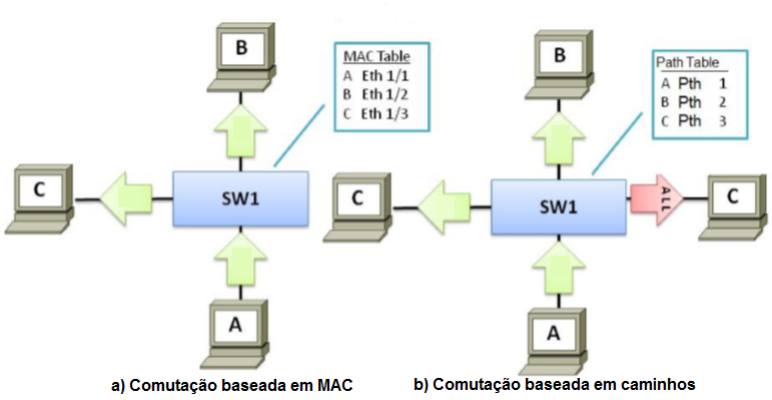


Figura 3. Comportamento de aprendizagem

O *switch* Ethernet tradicional (a) cria e mantém a tabela MAC que relaciona, para cada MAC conhecido pelo *switch*, uma porta de saída. O aprendizado inicial dos MACs se dá por inspeção dos quadros que transpassam o *switch*, verificando o campo MAC de origem de um tráfego entrante e relacionando-o à porta de onde foi recebido esse tráfego. O processo de aprendizagem é realizado por cada *switch* de forma isolada. Diferentemente, um *switch* de comutação baseada em caminhos (b) relaciona cada MAC a um caminho que deve ser tomado para comutar o pacote. Em uma abordagem SDN, o processo de aprendizagem é regido centralmente por um controlador SDN, que gerencia todos os *switches* da infraestrutura de rede.

Uma vantagem da comutação baseada em caminhos é a possibilidade de agregação de rotas, que ocorre quando dois ou mais *hosts* se encontram no mesmo *switch* destino. Desta forma, apenas uma entrada na tabela de comutação em *switches* intermediários é suficiente para a comutação dos dados que utilizam o mesmo caminho. No caso em que outro fluxo de comunicação é estabelecido entre os mesmo *switches* de origem e destino, é utilizado o mesmo *path* estabelecido.

4.1.2. Operação do PathFlow

A Figura 4 ilustra a operação do *switch* em uma solução PathFlow. A partir do *start-up* da rede, os *switches* encaminham ao controlador o *status* de suas conexões e dos *hosts* diretamente ligados, para que esse crie o mapeamento de toda a rede. Por outro lado, o

controlador envia mensagens para a configuração da tabela de fluxo de cada *switch* para que esse possa descobrir como comutar um pacote entrante. Internamente, o controlador cria e mantém um conjunto de tabelas que refletem o estado atual de todos os *switches* e de toda infraestrutura de rede.

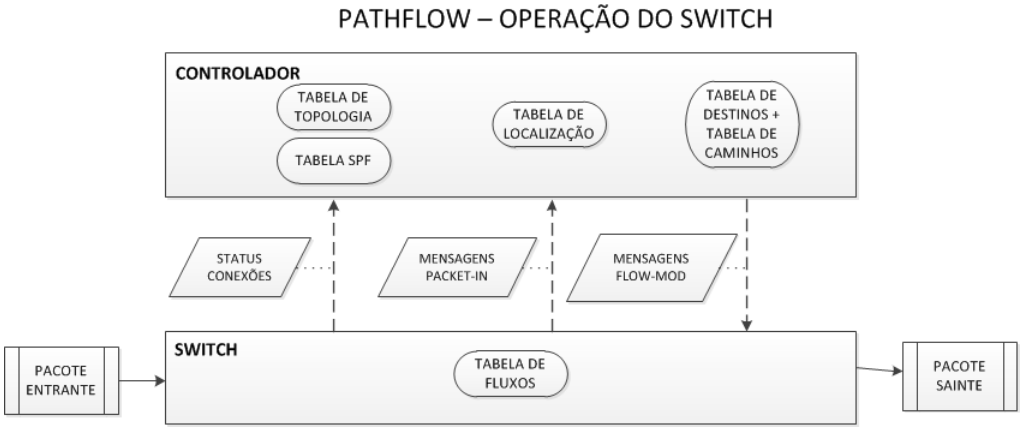


Figura 4. Operação do Switch

Em relação a um *switch* Ethernet tradicional, o aprendizado inicial não é modificado. Os *switches* de borda *Top of Rack* (ToR), que recebem diretamente os servidores, podem, através de monitoração do tráfego inicial ou através de ARP gratuitos (*gratuitous* ARP), conhecer quais MACs estão diretamente ligados em suas portas. A diferença é que, em se tratando de uma infraestrutura SDN, essa informação deve ser repassada ao controlador SDN para que esse atualize o mapeamento da rede. Disso, segue a operação inicial dos *switches*:

1. O controlador SDN, no *start-up* dos *switches*, monta uma tabela de topologia (*topology table*) dos *switches* e calcula os caminhos (*paths*) para que todos os *switches* possam alcançar todos os demais.
   (a) Os caminhos são calculados através de algoritmo de *shortest path first* (SPF), baseado na velocidade dos *links* ou quantidade de saltos;
   (b) Os *paths* são unidirecionais; e
   (c) Aos *paths* são atribuídos ***tags***, que pode ser de 6 ou 12 bits, a depender da implementação.
2. Os *switches*, por sua vez, começam a montar sua tabela de destinos (*destination table*) relacionando os seus MACs locais – máquinas diretamente ligadas – às suas portas de saída.
   (a) A descoberta dos *hosts* locais pode ocorrer tanto com a verificação do tráfego entrante nas portas como pela emissão de ARPs gratuitos; e
   (b) Todos os *switches* devem informar ao controlador SDN o conteúdo dessa tabela para que esse crie a tabela de localização (*localization table*) geral da rede, relacionando em qual *switch* e porta e VLAN estão associados todos os MACs da rede.
3. Quaisquer alterações na *destination table* precisa ser notificada ao controlador SDN para que esse atualize a *localization table*.
4. Pode haver na *destination table* mais de um caminho para o mesmo destino. Nesse caso, são instaladas as múltiplas entradas e é habilitado o balanceamento de carga.

5. Concomitamente, o controlador SDN informa aos *switches* os caminhos (*paths*) para os demais *switches*. Os *switches* criam uma tabela de caminhos (*path table*) associando os caminhos à porta de saída.

Após a operação inicial, segue o procedimento em caso de migração de algum host. Primeiramente, existem duas formas de atualização da localização do *host*, a autônoma e a *triggered*:

1. A atualização autônoma ocorre quando o controlador SDN "percebe" a alteração de um *host* devido à atualização da *localization table* que é informada por outro *switch*. Nesse caso:
   (a) O controlador SDN atualiza sua *localization table*; e
   (b) Envia uma atualização para que todos os *switches* façam a troca em suas *destination table* e *path table* para o novo caminho, o que faz a comutação ser instantaneamente alterada.
2. A atualização *triggered*, diferentemente da autônoma, ocorre quando o controlador é avisado de uma mudança - por exemplo, após ter sido recebida uma diretiva do hipervisor. Todos os demais procedimentos são iguais aos da atualização autônoma.

Essa proposta permite atender o requisito principal de migração de *hosts* entre redes distintas, mantendo-se o endereço IP e as conexões a ele. Ao mesmo tempo, propicia uma comutação que permite uma maior escalabilidade da rede. A arquitetura do PathFlow está detalhada no diagrama em blocos ilustrado na Figura 5.

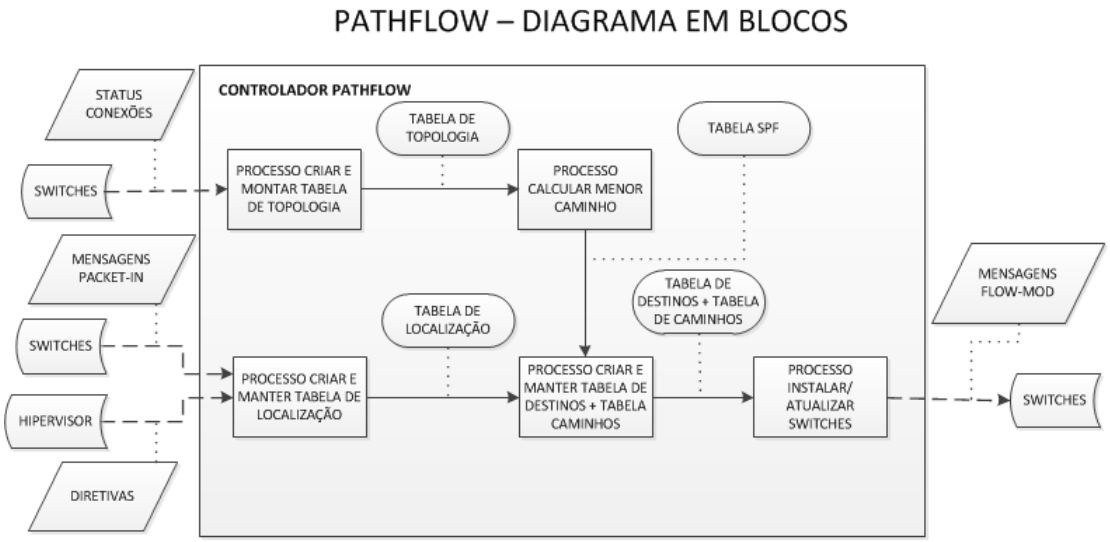


Figura 5. Diagrama em blocos do PathFlow

4.2. Implementação do PathFlow

O PathFlow v1.0 foi desenvolvido em Python como componente do controlador POX [POX 2015], sendo compatível com a versão 1.0 do Openflow assim como o próprio controlador POX. O PathFlow utiliza outros componentes do controlador POX. O componente **openflow.discovery** é utilizado pelo processo **Criar e manter tabela de topologia**,

para o recebimento de mensagens LLDP dos *switches* da infraestrutura e criação da tabela de topologia. Esse processo também utiliza o componente **openflow.spanning_tree** para contrução de árvore spanning-tree livre de *loops*. O componente **pox.lib.revent** é utilizado por vários processos para ”escutar”eventos dos *switches*, como **Packet-In**, **PortStatus** e **ConnectionUp**. Como a versão 1.0 do OpenFlow não possui campo de *label* MPLS na tabela de fluxo, foi usado o campo IP ToS (Tabela 1) para implementar o *tag* gerado para cada caminho.

5. Análise Experimental

5.1. Ambiente de emulação

O PathFlow foi avaliado em um ambiente emulado através da ferramenta Mininet [Lantz et al. 2010]. O ambiente de emulado foi executado em um *laptop* com sistema operacional Windows®8 e processador Intel®i7 64 bits com 8,00 GB de RAM. Acima do sistema operacional foi instalado um VMWare®Workstation 10 em que foram criadas duas máquinas virtuais com Ubuntu 14.04 LTS: a primeira para execução do Mininet 2.2.1 associado ao OVS 2.3.1 e outra para execução exclusiva do controlador Pox 0.2.0.

5.2. Etapa de verificação do funcionamento do sistema

A primeira etapa foi analisar a funcionalidade de migração do sistema. Os testes de mobilidade foram realizados fazendo a desconexão de estações virtuais no Mininet, reproduzindo o equivalente a um *handoff* de *hosts* ligados aos *switches* OVS. Os testes de migração de VMs são aproximações que envolvem um *handoff*, dado que esse envolve um conjunto de troca de sinalizações entre estado anterior e posterior para que as conexões se mantenham após a migração. Para os testes foi utilizada a topologia constante na Figura 6. Trata-se de uma topologia com um núcleo (Switch3) e três *switches* como acesso (Switch1, Switch2 e Switch4).

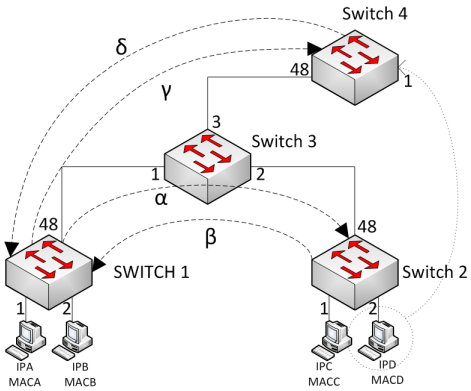


Figura 6. Verificação do funcionamento do sistema

Será indicado a seguir o comportamento da rede **pós migração**. O *hand-off* ocorreu com o *host* **D**, que migrou do Switch2 para o Switch4. Foi utilizado o *script* **mobility.py**, pertencente ao pacote Mininet, para esse teste. Esse *script* utiliza extensões do *switch* OVS para desassociar/associar (*detach/attach*) *hosts* às suas portas. A migração ocorreu no caso autônomo, em que o Controlador “percebe” a migração através de inspeção de mensagem de **PortStatus-event.deleted** e **Portstatus-event.added** geradas

quando um *host*, originalmente associado ao Switch3, deixa de ser conectar a este *switch* e “aparece” no Switch4.

O teste foi realizado efetuando-se um *ping* contínuo com intervalo de 0,01 segundos de todas as máquinas para a estação móvel, assim como da estação móvel para outra estação. A migração ocorreu tendo sido perdidos cinco pacotes de *ping*, o que significa que a migração neste cenário virtual ocorreu em apenas 0,05 segundos, aproximadamente.

O PathFlow possibilita a livre movimentação de uma máquina virtual para quaisquer pontos da rede gerenciada pelo sistema. Isso é possível porque as informações que possibilitam a segregação em sub-redes - tipicamente VLANs em *Data Centers* TCP/IP - não estão registradas de forma autônoma e pulverizada nos *switches* da infraestrutura, mas sim centralizados no controlador SDN, permitindo que essa informação ”acompanhe”o *host* para onde esse for.

Uma das principais diferenças do PathFlow em relação ao VICTOR [Hao et al. 2009], nesse ponto, é que o VICTOR utiliza tabelas de localização relacionando a tupla {Switch, Porta} com o IP, ao invés do MAC como no PathFlow. Uma vantagem da utilização do MAC é que a descoberta pode ocorrer sem precisar escalonar ao nível de IP. Além disso, não seria possível descobrir o IP em comunicações que não fossem relacionados a esse protocolo.

5.3. Etapa de avaliação da escalabilidade do sistema

Na etapa da avaliação, para uma mesma topologia será registrada a quantidade de entradas na tabela de comutação para três diferentes arquiteturas: um *Data Center* convencional TCP/IP baseado em comutação de pacotes; um *Data Center* SDN tradicional baseado na comutação por fluxos, de acordo com a arquitetura de VICTOR, e o PathFlow. Foi utilizada a arquitetura *spine and leaf* [Greenberg et al. 2008] para avaliação das tabelas de comutação, devido a mesma ser amplamente utilizada em *Data Centers*. Foram utilizados dois *switches* na função de *spine* e quatro *switches* na função *leaf*, interconectados conforme a Figura 7.

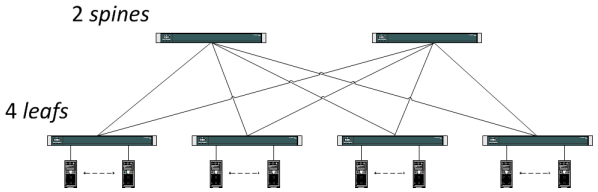
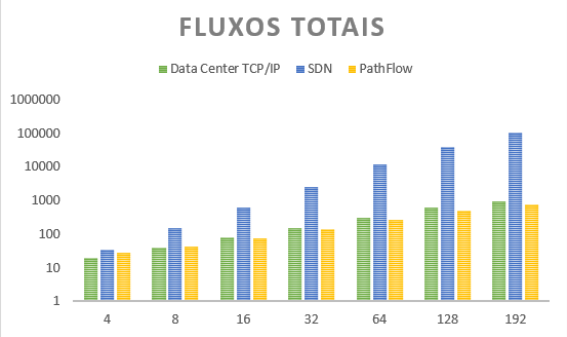
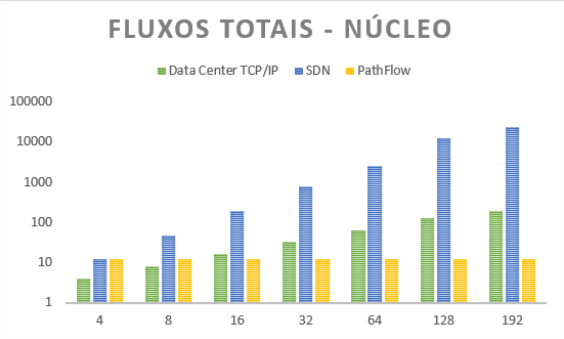


Figura 7. Topologia Spine and Leaf utilizada na avaliação de tamanho de tabelas

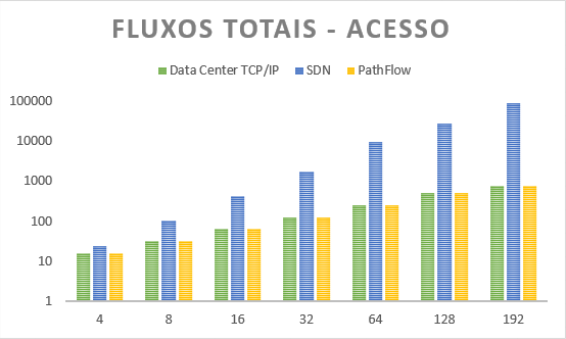
No gráfico da Figura 8(a), estão registrados os dados obtidos para a quantidade de fluxos gerais do sistema. Já era esperada uma grande diferença de uma solução SDN para uma solução de *Data Center* convencional baseada em TCP/IP, pois uma solução SDN tradicional gera uma entrada para cada fluxo de comunicação - que nesse caso se restringiu a um simples **ping** - ocasionando um crescimento praticamente exponencial. Porém, o PathFlow consegue diminuir a quantidade total de fluxos por conta da agregação de caminhos que faz com que, a partir de 16 *hosts*, já se consiga ter uma quantidade menor de fluxos se comparado ao *Data Center* TCP/IP, que possui um crescimento praticamente linear em relação à quantidade de *hosts*.



((a)) Fluxos Totais por quantidade de *hosts*



((b)) Fluxos Totais por quantidade de *hosts* - Núcleo



((c)) Fluxos Totais por quantidade de *hosts* - Acesso

Figura 8. Fluxos por quantidade de *hosts*

Fazendo-se uma análise por camada, no Núcleo da infraestrutura, explicitado no gráfico da Figura 8(b), também é evidente o comportamento exponencial do SDN tradicional. Da mesma forma, fica evidente a agregação do PathFlow, que possui uma quantidade de fluxos fixa baseada apenas na topologia da rede. Enquanto isso, é evidenciada a linearidade do *Data Center* TCP/IP, cuja tabela depende apenas da quantidade total de *hosts*. Em relação à camada de Acesso, ilustrado no gráfico da Figura 8(c), verifica-se a relação linear do número de fluxos à quantidade de *hosts* da infraestrutura, comparando-se o PathFlow ao *Data Center* TCP/IP, com a solução SDN tradicional mantendo seu comportamento exponencial.

Analisando-se os dados obtidos, é notado que, embora o PathFlow seja uma solução SDN e realize suas comutações baseadas em fluxos, não acompanha o crescimento exponencial de uma solução SDN tradicional e possui sua escalabilidade compatível com a de um *Data Center* tradicional TCP/IP, porém ganhando do mesmo quando verificada a escalabilidade da solução nos *switches* de Núcleo.

6. Conclusão e Trabalhos Futuros

Este trabalho endereça o problema de migração *on-line* de VMs entre redes distintas em *Data Centers* tomando como base recentes trabalhos de virtualização de redes. Propõe-se para tal uma **solução SDN de comutação baseada em caminhos** chamada **PathFlow**.

O PathFlow utiliza como base o fato de que o controlador SDN possui uma visão holística da rede, conhecendo todos os dispositivos de encaminhamento (FE) e ligações entre eles, conseguindo dessa forma criar e manter uma topologia e calcular caminhos

mínimos. Desta forma, o PathFlow possibilita a livre movimentação de uma máquina virtual para quaisquer pontos da rede gerenciada pelo sistema, já que as informações que possibilitam a segregação em sub-redes não estão registradas de forma autônoma e pulverizada nos *switches* da infraestrutura, mas sim centralizados no controlador SDN.

O PathFlow possui a capacidade de agregar caminhos em um mesmo *path*, permitindo dessa forma que o crescimento das tabelas de comutação seja linear com o crescimento de *hosts* na camada de acesso, e dependente apenas da topologia do *Data Center* no núcleo. Portanto, embora o PathFlow seja uma solução SDN, este não acompanha o crescimento exponencial de uma solução SDN tradicional e possui sua escalabilidade compatível com a de um *Data Center* tradicional TCP/IP.

Como sugestões de trabalhos futuros, indica-se a utilização de controladoras com suporte ao Openflow 1.1.0 ou acima, a partir do qual podem ser implementadas **tags** de maior capacidade compatíveis com o MPLS, assim como suporte ao recebimentos de diretivas dos hipervisores para o sincronização de eventos e consequentemente menor tempo de migração.

Agradecimentos

Agradecemos o apoio recebido da FAPERJ no contexto do projeto de pesquisa E-26/203.446/2015.

Referências

Anderson, T., Peterson, L., Shenker, S., and Turner, J. (2005). Overcoming the internet impasse through virtualization. *Computer*, 38(4):34–41.

Bari, M., Boutaba, R., Esteves, R., Granville, L., Podlesny, M., Rabbani, M., Zhang, Q., and Zhani, M. (2013). Data center network virtualization: A survey. *Communications Surveys Tutorials, IEEE*, 15(2):909–928.

Chowdhury, N. M. K. and Boutaba, R. (2010). A survey of network virtualization. *Computer Networks*, 54(5):862 – 876.

CISCO (2015). Cisco ios local-area mobility. `http://www.cisco.com/en/US/products/ps9390/products_white_paper09186a00800a3ca5.shtml`. [Online; acessado em 02-março-2015].

Davie, B. and Gross, J. (2013). A stateless transport tunneling protocol for network virtualization (stt). *draft-davie-stt-04*.

Farinacci, D., Maino, F., Ermagan, V., Hertoghs, Y., and Smith, M. (2013). Lisp control plane for network virtualization overlays. Online; acessado em 15-julho-2015.

Floyd, S., Handley, M., and Kohler, E. (2006). Ietf rfc 4340: Datagram congestion control protocol (dccp). *https://tools.ietf.org/html/rfc4340*.

Greenberg, A., Hamilton, J., Maltz, D. A., and Patel, P. (2008). The cost of a cloud: Research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73.

Gross, J., Sridhar, T., Garg, P., Wright, C., and Ganga, I. (2014). Geneve: Generic network virtualization encapsulation. *Internet Engineering Task Force, Internet Draft*.

Hao, F., Lakshman, T. V., Mukherjee, S., and Song, H. (2009). Enhancing dynamic cloud-based services using network virtualization. In *Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, VISA '09, pages 37–44, New York, NY, USA. ACM.

Johnson, D. (2004). Ietf rfc 3775: Mobility support in ipv6. *https://www.ietf.org/rfc/rfc3775.txt*.

Kim, H. and Feamster, N. (2013). Improving network management with software defined networking. *Communications Magazine, IEEE*, 51(2):114–119.

Lantz, B., Heller, B., and McKeown, N. (2010). A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA. ACM.

Lara, A., Kolasani, A., and Ramamurthy, B. (2014). Network innovation using openflow: A survey. *Communications Surveys Tutorials, IEEE*, 16(1):493–512.

Maltz, D., Bhagwat, P., et al. (1998). Msocks: An architecture for transport layer mobility. In *INFOCOM'98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1037–1045. IEEE.

Matsumoto, A., Kozuka, M., Fujikawa, K., and Okabe, Y. (2003). Tcp multi-home options. *draft-arifumi-tcp-mh-00. txt, IETF Internet draft*.

McKeown, N. (2011). How sdn will shape networking. Online; acessado em 22-fevereiro-2015.

ONF (2015). Open network foundation. `https://www.opennetworking.org/`. [Online; acessado em 15-abril-2015].

Perkins, C. (2002). Ietf rfc 3344: Ip mobility support for ipv4. *http://www. ietf. org/rfc/rfc3344.txt*.

POX (2015). About pox. `http://www.noxrepo.org/pox/about-pox/`. [Online; acessado em 15-fevereiro-2015].

Shenker, S., Casado, M., Koponen, T., McKeown, N., et al. (2011). The future of networking, and the past of protocols. *Open Networking Summit*, 20.

Snoeren, A. C. (2002). *A session-based approach to Internet mobility*. PhD thesis, PhD Thesis, Massachusetts Institute of Technology.

Sridhar, T., Kreeger, L., Dutt, D., Wright, C., Bursell, M., Mahalingam, M., Agarwal, P., and Duda, K. (2013). Vxlan: a framework for overlaying virtualized layer 2 networks over layer 3 networks. *draft-mahalingamdutt-dcops-vxlan-04*.

Sridharan, M., Greenberg, A., Venkataramiah, N., Wang, Y., Duda, K., Ganga, I., Lin, G., Pearson, M., Thaler, P., and Tumuluri, C. (2011). Nvgre: Network virtualization using generic routing encapsulation. *IETF draft*.

Stewart, R. (2007). Ietf rfc 4960: Stream control transmission protocol. *https://tools.ietf.org/html/rfc4960*.