# Um Estudo Comparativo de Softwares de Roteamento para Uso em Redes Definidas por Software

**Luiz Fernando T. de Farias**[1], **Morganna Carmem Diniz**[1], **Sidney Cunha de Lucena**[1], **Carlos Nilton Araújo Corrêa**[1]

[1] Universidade Federal do Estado do Rio de Janeiro (UNIRIO), Rio de Janeiro – Brasil

***Resumo.*** *Redes Definidas por Software (SDN) possibilitam a virtualização de diferentes topologias e aplicações de rede, com objetivo de teste ou produção, executando-as sobre uma infraestrutura compartilhada. Neste contexto, uma questão importante é a diferença entre o nível de serviço provido por diferentes aplicações de roteamento sobre ambientes virtualizados baseados em SDN. Este artigo fornece uma análise comparativa do desempenho de diferentes suítes de roteamento voltadas para uma arquitetura de roteamento virtualizado sobre redes OpenFlow. Para tal, foram testados os softwares de roteamento BIRD, Quagga e XORP em diferentes topologias e graus de conectividade. Medidas como tempos de distribuição de rotas e de convergência, ou consumo de CPU, foram obtidas de maneira a apoiar decisões quanto ao uso destas plataformas.*

***Abstract.*** *Software Defined Networks (SDN) allow for the virtualization of different network topologies and applications, either for experimental or production purposes, running over a shared infrastructure. In this context, an important question that arises is about the different service levels provided by different routing applications over virtualized environments based on SDN. This paper provides a comparative analysis of the performance of different routing suites used for virtualized routing architectures over OpenFlow networks. Therefore, the open source routing platforms BIRD, Quagga and XORP were evaluated for different topologies and connectivity levels. Measures like route distribution and convergence times, or CPU utilization, were taken in order to support tactical decision related to the use of such platforms.*

## 1. Introdução

No ambiente tradicional, são usados roteadores implementados com *softwares* e *hardwares* proprietários. Uma opção é o uso de suítes de roteamento (SRs) que são *softwares* que podem ser instalados em máquinas PC ou estar embarcado num hardware de *switch* com o Linux, por exemplo, fazendo-o funcionar como um roteador. Em [He 2005], Liwen He defende a utilização de *softwares* programáveis quando nos informa que o volume de tráfego na Internet dobra a cada 12 meses, sendo isto superior à capacidade dos fabricantes de produzir *hardware* que atenda a demanda. A mesma opinião é defendida em [Feamster et al. 2004], onde são apresentadas as limitações de escalabilidade dos atuais protocolos de roteamento.

A virtualização de redes permite particionar a capacidade dos componentes de uma rede física tornando possível estabelecer múltiplas infraestruturas lógicas distintas sobre os mesmos componentes [Corrêa et al. 2012]. Novas arquiteturas de rede com

foco de uso em *Datacenters* e *Cloud Computing* impulsionam a adoção de técnicas de virtualização [Rothenberg et al. 2012] por oferecerem flexibilidade para realocação de recursos físicos. No entanto, [Carrissimi 2008] salienta o problema da migração de máquinas virtuais entre diferentes redes o que é uma premissa para estas tecnologias que necessitam alterar a topologia da rede sob demanda. As topologias empregadas neste trabalho fazem o levantamento de métricas que consideram os recursos computacionais do servidor e o tempo para refazer a topologia utilizada em ambiente virtual.

Outro aspecto importante em relação às SRs são as redes definidas por *software* (SDN, do inglês *Software-Defined Networks*) que permitem implementar a separação das funções de encaminhamento de uma rede em planos de dados (*hardware*) e controle (*software*). Por exemplo, os protocolos como OSPF e BGP podem ser implementados pelas SRs no plano de controle. Essas redes podem ser usadas tanto em ambiente de produção como em ambiente de testes, e mesmo em ambiente com as duas finalidades. Portanto, uma das vantagens desta tecnologia é a facilidade de instalação e de migração entre diferentes ambientes.

As SRs disponibilizadas com licença de *software* livre propiciam vantagem quanto à possibilidade da programação de seus módulos. O uso de *softwares* roteadores de código aberto se tornou uma alternativa interessante, pois tanto o *software* quanto o *hardware* possuem documentação e estrutura bem conhecidas. Além disso, a utilização dessa solução é uma opção de menor custo podendo alcançar alto desempenho e ser uma vantagem competitiva para as empresas com baixo orçamento [Hagsand et al. 2008].

Os *softwares* de roteamento implementados em máquinas PC, por exemplo, também podem trabalhar em conjunto com os roteadores tradicionais, o que permite realizar testes em ambientes de produção. Isto significa que não há necessidade em se adotar uma única tecnologia. É possível inclusive implementar esses *softwares* de roteamento em ambiente virtual possibilitando o compartilhamento dos recursos computacionais e a reconfiguração da topologia na presença de falhas.

A demanda por novos serviços tem sido crescente desde o surgimento da Internet, o que produziu o aumento na complexidade da sua infraestrutura [Feamster et al. 2004] e fez surgir a necessidade por novas soluções em *hardware* e *software*. O estudo necessário para a criação dessas soluções tem sido realizado principalmente com bancos de ensaio de rede (*testbed*) e simuladores como o NS-2 [Handley et al. 2002]. O uso das SRs permite que infraestruturas de redes possam ser usadas para a criação de ambientes de experimentação que se comportem de forma mais próximas das condições que são observadas em situação real.

O potencial das redes definidas por *software* pode ser explorado através do uso do protocolo OpenFlow que fornece uma interface lógica de programação de código aberto para dispositivos comutadores de rede. Um exemplo de uma aplicação de roteamento IP que faz uso do protocolo OpenFlow é o RouteFlow [Nascimento et al. 2011]. O RouteFlow [Nascimento et al. 2011, Rothenberg et al. 2012] propõe o gerenciamento dos comutadores de rede utilizando SRs e virtualização. No RouteFlow, os switches OpenFlow são associados a roteadores virtuais e entre eles são estabelecidadas as conectividades virtuais conforme a topologia física subjacente. Sempre que as tabelas de encaminhamento dos roteadores virtuais são atualizadas, o plano de gerenciamento do RouteFlow auto-

maticamente atualiza as tabelas de fluxo dos respectivos switches OpenFlow. Os elementos do plano de controle responsáveis pela implementação dos protocolos de roteameto são formados por ambientes virtuais na forma de contêineres. O uso de contêineres proporciona simplicidade no gerênciamento dos protocolos de roteamento e escalabilidade do plano de controle, ao serem acrescentas MVs de acordo com o surgimento de novas topologias lógicas. A comparação das SR's em ambiente virtualizado permite observar o comportamento destes *softwares* como elementos do plano de controle e avaliar as características das arquiteturas mais interessantes de serem aproveitadas ou alteradas no desenvolvimento de soluções para SDN.

Este trabalho tem por objetivo apresentar uma análise comparativa do desempenho de diferentes SRs voltadas para uma arquitetura de roteamento virtualizado baseada em contêineres. Para tal, foram testados os *softwares* de roteamento BIRD, Quagga e XORP em diferentes topologias, graus de conectividade e uso dos protocolos OSPF e BGP. Medidas como tempos de distribuição de rotas e de convergência, além de consumo de recursos, foram obtidas de maneira a apoiar decisões quanto ao uso destas plataformas. Considerando os atuais projetos de implementações SDN, por exemplo o RouteFlow, os experimentos foram realizados utilizando um único host. Os recursos de *hardware* são compartilhados entre as suítes de roteamento.

O restante deste artigo é organizado da seguinte forma. Na seção 2, é feita a descrição das SRs utilizadas nos experimentos, enquanto o ambiente de experimentação é discutido na Seção 3. Os resultados obtidos são apresentados na Seção 4, e na Seção 5 são apresentadas as conclusões.

## 2. Suítes de Roteamento

As suítes de roteamento são *softwares* que preenchem de forma automática as tabelas de roteamento usadas pelo *kernel* do sistema operacional para o encaminhamento dos pacotes de dados. Com esta finalidade, as SRs implementam protocolos de roteamento e disponibilizam ferramentas para a gerência da rede e interface de console para controle do ambiente. Para se ter uma ideia da importância dos *softwares* de roteamento no ambiente da Internet, basta lembrar que os IXPs (*Internet eXchange Provider*) da Europa possuem 66 *route servers* onde 37 deles usam SRs para trocar tabelas de roteamento: 15 com BIRD, 8 com Quagga, 8 com OpenBGP e 6 com outras SRs [Sanghani 2012].

Existem dois tipos de arquitetura para as SRs [Kaliszan et al. 2012]. No primeiro tipo, múltiplos processos são organizados em três camadas [Handley et al. 2002]: uma camada que faz parte do *kernel* do sistema operacional, uma camada formada pelos processos que implementam as funcionalidades dos protocolos de roteamento e uma camada com os processos responsáveis pelo gerenciamento das SRs. A Figura 1 mostra este tipo de arquitetura, onde a segunda camada apresenta os protocolos de roteamento RIP, OSPF e BGP. O segundo tipo de arquitetura é o monolítico. Neste caso, existe um único arquivo executável composto por módulos estaticamente ligados [Trosvik 2006]. A cada inclusão, alteração ou deleção de módulos, um novo arquivo executável é gerado. Por exemplo, pode-se definir um módulo para cada protocolo de roteamento disponível na SR.

Neste trabalho são utilizadas nos testes as SRs de código aberto BIRD, Quagga e XORP em máquinas virtuais. O BIRD possui arquitetura monolítica, enquanto o Quagga

e o XORP possuem arquitetura de três camadas. A utilização do ambiente virtual possui a vantagem de permitir que topologias sejam criadas e reconfiguradas de forma mais rápida e simplificada que as instalações de *softwares* de roteamento feitas diretamente sobre sistemas operacionais hospedeiros [Kaliszan et al. 2012].

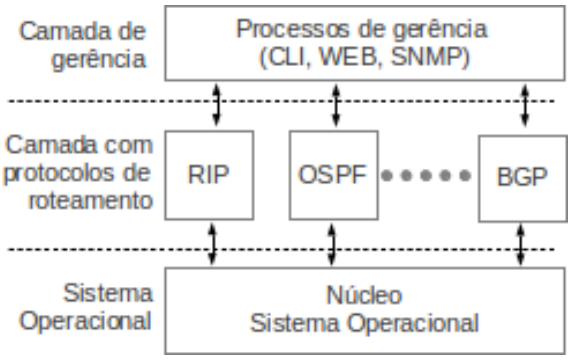


**Figura 1. Arquitetura multiprocessos**

A virtualização de sistemas [Bhatia et al. 2008] possui três formas de implementação: completa, paravirtualização e baseada em contêineres. No primeiro caso, cada ambiente tem total controle sobre o que vai executar, escolhendo inclusive o seu *kernel* de sistema operacional. Quando há necessidade de alocação de algum recurso de *hardware*, o pedido é recebido pelo virtualizador que o traduz e o passa para o sistema operacional da máquina. O segundo caso é considerado uma variação do anterior [Bhatia et al. 2008] provendo melhores resultados através da otimização dos mecanismos de emulação do *hardware*. Para este tipo de virtualização o sistema operacional que será instalado na MV tem que sofrer alterações. No terceiro caso, uma imagem do sistema operacional virtualizado é compartilhada entre os diversos ambientes, ou seja, todos os ambientes usam um mesmo sistema operacional. Não há disputa de recursos de *hardware* entre os contêineres, já que os recursos de *hardware* são divididos durante a criação dos ambientes. Segundo [Corrêa et al. 2011], as técnicas de virtualização de sistemas operacionais baseadas em contêineres chegam a uma performance até duas vezes superior às outras técnicas, principalmente em cenários de serviços baseados em *web*. Além disso, a técnica de virtualização baseada em contêineres apresenta um melhor desempenho quando se demanda mais máquinas virtuais rodando simultaneamente.

### 2.1. BIRD (Internet Routing Daemon)

O projeto BIRD [Ondrej et al. 2012] foi iniciado em 1999 e é o *software* mais utilizado como centralizador e distribuidor de rotas anunciadas (*route server*) nos pontos de troca de tráfego (PTT) na Europa [Sanghani 2012]. Os módulos do BIRD oferecem serviços como gerenciamento dos recursos compartilhados, ligação com o *kernel* do sistema operacional e gerenciamento de múltiplas tabelas de roteamento. Esta SR permite a configuração de uma tabela primária e de várias tabelas secundárias. O BIRD utiliza o conteúdo da tabela primária como fonte das rotas passadas para o *kernel* do sistema operacional. As tabelas secundárias são utilizadas quando existem mais de um módulo de um mesmo protocolo de roteamento dentro de um contêiner. Neste caso, o BIRD usa um protocolo próprio chamado *pipe* que permite a troca de informações de encaminhamento entre as tabelas.

Nos experimentos realizados neste trabalho, como apenas um protocolo de roteamento é usado por vez, não há necessidade de tabelas secundárias e nem do uso do protocolo *pipe*.

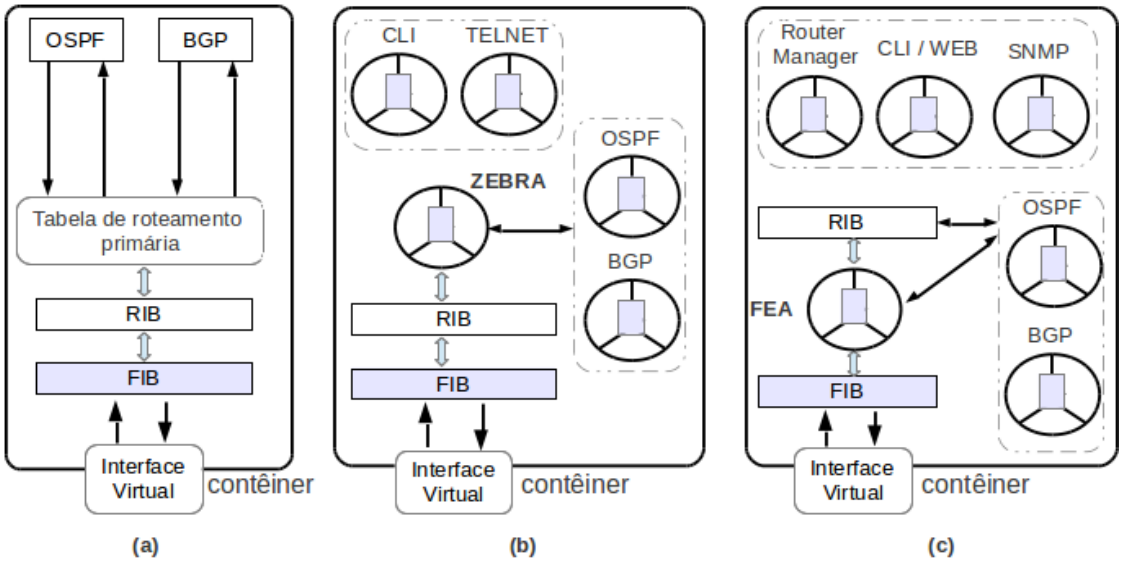


**Figura 2. Contêineres com o (a) BIRD (b) Quagga e (c) XORP**

A Figura 2a mostra uma possível configuração para o BIRD no ambiente de máquinas virtuais usando contêineres. Neste exemplo, existem dois protocolos de roteamento que alimentam a tabela primária com as rotas calculadas. Cada contêiner possui duas bases de dados do *kernel* do sistema operacional: RIB (*Routing Information Base*) e FIB (*Forwarding Information Base*). A RIB é preenchida automaticamente com o conteúdo da tabela primária ou de forma manual pelo administrador do sistema. A FIB utiliza as rotas selecionadas pela RIB para encaminhar os pacotes de dados. Portanto, é possível construir um ambiente de redes onde cada contêiner é um roteador e a ligação entre os diversos roteadores é feita através de uma interface virtual.

## 2.2. Quagga

O Quagga [Ishiguro 2012] é um *software* de roteamento que pode atuar como "*router server*". O núcleo do Quagga é constituído de um processo de sistema chamado Zebra que atua como uma camada de abstração (interface) entre o *kernel* do Linux e os protocolos de roteamento. Portanto, para usar o Quagga em outros sistemas operacionais, além do Linux, basta alterar o módulo Zebra.

Os processos que compõem o Quagga são independentes entre si, pois a arquitetura do *software* não é monolítica como o BIRD. Isto significa que novos processos podem ser inicializados a qualquer momento, assim como processos em execução podem ser reconfigurados ou finalizados sem interferir nos outros processos. De forma similar ao BIRD, pode-se construir um ambiente de redes onde cada contêiner com Quagga é um roteador e a ligação entre os diversos roteadores é feita através de uma interface virtual.

A Figura 2(b) mostra uma implementação do Quagga com cinco processos: o Zebra, uma interface de linha de comando (CLI), um servidor de acesso remoto (Telnet) e dois processos que implementam os protocolos de roteamento OSPF e BGP.

### 2.3. XORP (eXtensible Open Router Plataform)

O XORP [Handley et al. 2002] é um *software* de roteamento que foi desenvolvido inicialmente para a pesquisa acadêmica, mas que atualmente é também utilizado em ambiente de produção. Ele atua como *route server* de forma equivalente ao BIRD e ao Quagga.

A Figura 2(c) mostra uma possível configuração do XORP que, como o Quagga, é composto por processos independentes. Neste exemplo, existem seis processos: um gerenciador dos recursos da suíte de roteamento (*router manager*), a interface de interação com o usuário (linha de comando (CLI) ou de gerenciamento pela *web*), um processo que implementa o protocolo SNMP de gerenciamento de redes, dois protocolos de roteamento (OSPF e BGP) e a FEA (*Forwarding Engine Abstraction*) que é responsável por gerenciar a RIB e a FIB.

Atualmente, o XORP é empregado como roteador da Internet de nova geração por várias empresas de solução de tecnologia como, por exemplo, Pica8 [Pica8 ] e Candela Technologies [Candelatech ].

## 3. Ambiente de Testes

Para os testes deste trabalho, foi utilizada uma máquina com processador Quad Core de 64 bits, 8 GBytes de memória e sistema GNU-Linux Debian 6.0.6 como o sistema operacional hospedeiro. O ambiente de testes foi criado com a máquina virtual LXC 0.7.2 no qual foram instaladas as suites de roteamento BIRD 1.3.4, QUAGGA 0.99.20 e XORP 1.8.3 em contêineres.

A conectividade que permite a troca de mensagens entre os contêineres é obtida através da associação da interface de rede local ethernet (ETH) e as interfaces virtuais (VETH - *Virtual Ethernet device*). Isto é feito através da configuração de uma ponte (*linux bridge*) no *kernel* do Linux da máquina real. Cada VETH tem associado um número ethernet (MAC *address*) e o número IP . Desta forma, cada contêiner corresponde a um roteador e consiste em uma cópia isolada da suíte de roteamento (Figura 3). Portanto, em um teste com uma grade de cem roteadores, existem cem contêineres virtuais.

Dois tipos de métricas são analisadas neste trabalho: cargas de processamento e tempos dos roteadores. No primeiro tipo, as métricas usadas foram propostas em [Corrêa et al. 2011]: o uso de CPU e o uso de memória em operação regular e em situação de falha. Para o segundo tipo, foram usadas as métricas propostas na RFC 4061, que trata do *Benchmarking* de implementações do protocolo OSPF [Manral et al. 2004]: tempo de convergência das tabelas de roteamento (tempo para que todos os roteadores conheçam a topologia da rede); tempo de convergência após falha (tempo para que todos os roteadores tomem conhecimento da mudança na topologia); e tempo médio necessário para conhecer a rota padrão (tempo médio para que o DUT conheça o melhor caminho para todas as sub-redes).

Para comparar o desempenho das suítes de roteamento com os protocolos OSPF e BGP, foi utilizada a medição interna (*white box*) [Manral et al. 2004] como descrita na RFC 4061. Neste método, umas das SRs é escolhida para a coleta das informações a serem analisadas e por isso é chamada de DUT (*device under test*). Embora o DUT seja penalizado pela inclusão do *software* responsável pela coleta dos dados, esse acréscimo no processamento é experimentado em todos os ambientes testados.

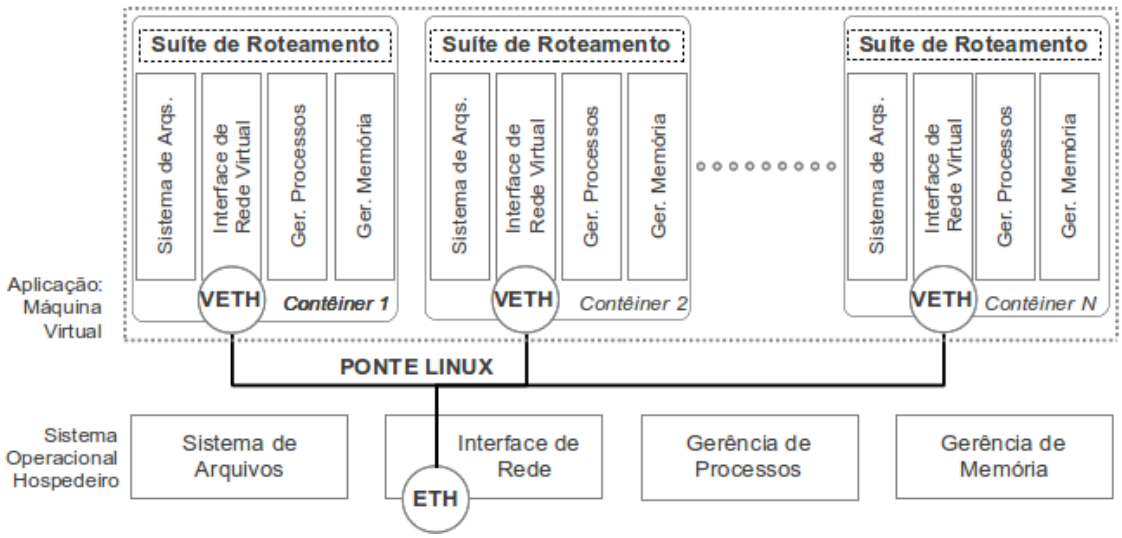


**Figura 3. Virtualização com contêineres.**

Os dados foram coletados a intervalos de 1 segundo e foi estudado o comportamento do sistema diante da ocorrência de falha em um dos roteadores. Uma falha ocorre quando uma adjacência ou um enlace deixa de funcionar [Manral et al. 2004]. As situações de falhas foram induzidas nos experimentos desligando uma SR, onde a distância entre o DUT e a SR desligada possui o maior número de saltos. Desta forma, a determinação do tempo que o DUT necessita para convergir equivale ao tempo de convergência [Bolla and Bruschi 2012] na topologia no caso do protocolo OSPF.

Para verificar a escalabilidade das suítes de roteamento com o protocolo OSPF, foram usadas grades variando de 2x2 (4 SRs) até 10x10 (100 SRs) de acordo com o modelo apresentado em [Bolla and Bruschi 2012]. A Figura 4a mostra a grade 3x3 usada nos testes identificando o DUT e o roteador com falha. Os valores utilizados na configuração do protocolo OSPF seguem as recomendações de [Manral et al. 2004]: o intervalo entre mensagens para os vizinhos informando de que ainda está ativo (*hello*) é 10 segundos; o tempo que aguarda mensagens do vizinho antes de considerá-lo inalcançável (*dead count*) é 40 segundos; o tempo de espera pela confirmação antes de fazer a retransmissão (*retransmit*) é 5 segundos; o tempo necessário para atualizar um dado na tabela de roteamento (*transmit delay*) é 1 segundo.

As medições realizadas com o protocolo BGP são as mesmas realizadas com o OSPF a diferença está na topologia utilizada. A Figura 4b mostra a topologia formada por três roteadores de borda que realizam anúncios das rotas de seus sistemas autônomos (AS) para o roteador (DUT) que atua na topologia como um PTT, permitindo a troca de anúncios de rotas entres os ASes.

Para verificar a escalabilidade, foram criados de 2 até 10 ASes onde cada um anuncia 1000 prefixos de forma similar aos testes feitos em [Jasinska and Malayter 2010]. Para representar o caso com falha um dos roteadores é desligado fazendo com que o AS deixe de anunciar as suas rotas.

Nos testes com o protocolo BGP, considerou-se que ASs não possuem qualquer política restritiva, ou seja, um pacote de dados pode atravessar qualquer AS para alcançar

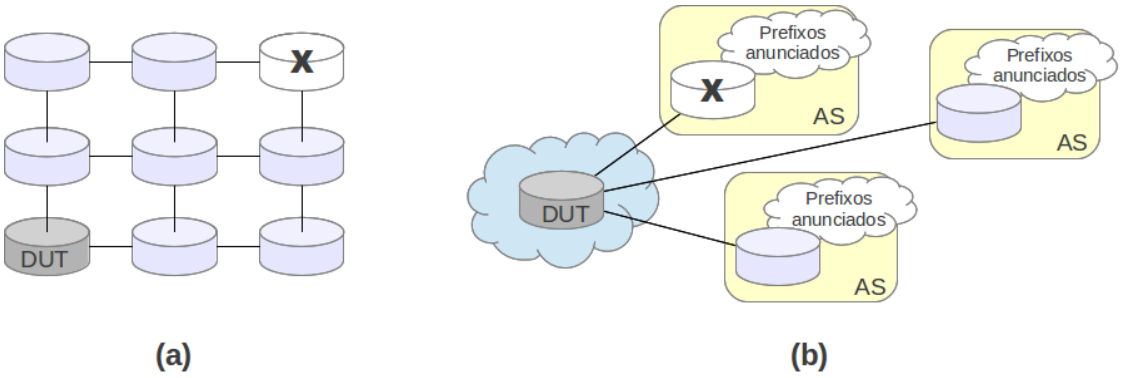


**Figura 4. banco de testes para (a) OSPF (b) BGP**

o seu destino.

Foram realizados 30 experimentos para cada topologia utilizando uma SR, totalizando assim 3.600 experimentos.

## 4. Análise dos Resultados

Esta seção discute os resultados obtidos com as SRs para os protocolos OSPF e BGP.

O OSPF é um protocolo usado para rotear pacotes dentro de um mesmo sistema autônomo (AS). Cada roteador OSPF constroi um mapa com a topologia da rede a partir das informações recebidas dos outros roteadores. Ao chegar um pacote, o seu endereço de destino IP é lido e a rota com menor custo (normalmente número de saltos) é escolhida. Quando uma falha ocorre na topologia, os roteadores propagam a informação e novas rotas são definidas para todos os possíveis destinos dentro da AS. Pacotes de dados com destino para redes fora da AS são enviados para o roteador de borda padrão ou para o roteador de borda mais próximo do destino.

O BGP é um protocolo utilizado para roteamento de pacotes de dados entre sistemas autônomos tendo como base regras não-técnicas (políticas) definidas por cada AS. Isto significa que é preciso um acordo prévio para que pacotes de dados atravessem diferentes ASs. Quando um roteador BGP anuncia as suas rotas, ele envia as informações que possui e recebe as tabelas de roteamento das outras ASs. Quando ocorre alguma mudança na topologia, apenas as alterações são propagadas.

### 4.1. OSPF

A Figura 5a mostra a utilização média da CPU nas diversas configurações de rede quando nenhuma falha é detectada na topologia, enquanto a Figura 5b mostra a utilização da CPU com a ocorrência de uma falha. Comparando os dois gráficos, é possível notar que a utilização da CPU do segundo é ligeiramente inferior a do primeiro. Isto é explicado pelo fato que existe um contêiner (suíte de roteamento) a menos no segundo caso. Por exemplo, na grade 9x9, no primeiro cenário o BIRD, o Quagga e o XORP apresentam, respectivamente, uma utilização de 46,20%, 48,05% e 51,99%; enquanto no segundo cenário a utilização dessas mesmas SRs reduz para 34,43%, 45,89% e 50,19%. A redução maior obtida com o BIRD se explica por este *software* conseguir se adaptar mais rapidamente à nova topologia que as outras duas SRs. Portanto, há dois fatores que influenciam

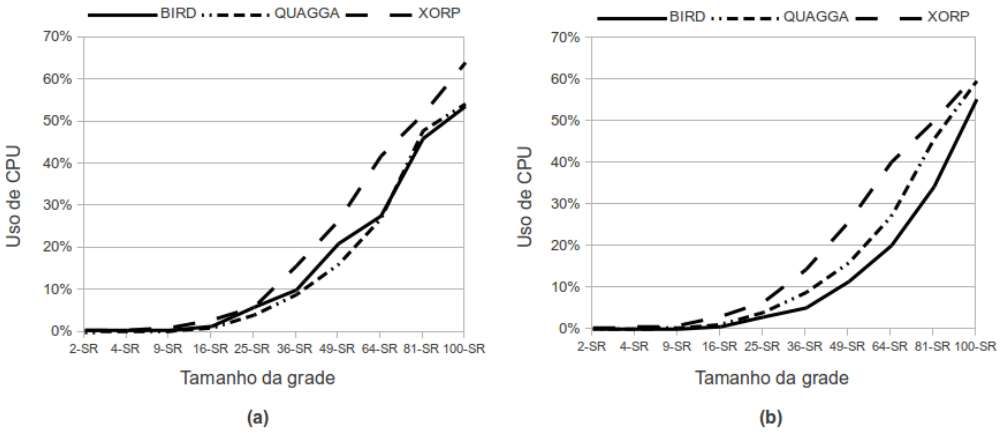


**Figura 5. Uso de CPU nos casos (a) sem falhas e (b) com falha**

o cálculo do valor: o menor gasto adicional de CPU para calcular as novas rotas; e uma menor quantidade de contêineres na topologia para consumir recursos. Ainda verificando os gráficos da Figura 5, o XORP apresenta um maior consumo de CPU quando comparado com as outras duas SRs para qualquer topologia. O BIRD possui a menor utilização de recursos no caso de falha por ter uma reconfiguração mais rápida. Quando não há falha, o Quagga gasta normalmente menos CPU que o BIRD. Por exemplo, na grade 7x7, o gasto é, em média 20%, menor. Entretanto, a diferença no consumo total de CPU das duas SRs, considerando qualquer topologia, é menor que 0,7%.

A Figura 6a mostra a utilização média da memória quando nenhuma falha ocorre, enquanto a Figura 6b mostra a utilização da memória com a ocorrência de uma falha. Na topologia sem falhas, a utilização máxima do BIRD é 94,68% na grade 6x6, a do Quagga é 97,77% na grade 4x4 e a do XORP é 98,34% na grade 5x5. Nos experimentos com falhas, a média de utilização em todas as grades apresenta valores equivalentes: BIRD com 75,23%, Quagga com 83,34% e XORP com 84,41%. Normalmente, o BIRD apresentou um menor consumo de memória quando comparado às outras duas SRs. Isto é interessante, pois o BIRD é um único arquivo executável (arquitetura monolítica) e, portanto, deveria apresentar um maior consumo de memória. Mas é preciso lembrar que o uso de memória virtual faz com que o sistema operacional selecione as partes do arquivo que são carregadas na memória principal. No Quagga e no XORP, o usuário especifica quais módulos devem ser carregados e o sistema operacional se responsabiliza por definir como a memória principal vai ser ocupada e como a comunicação entre os módulos será feita. Logo, é possível que o BIRD apresente, na maioria das vezes, um melhor desempenho devido a sua simplicidade de implementação. Também observa-se na Figura 6 que o con-

| **Memória** | 2-SR | 9-SR | 16-SR | 25-SR | 36-SR | 49-SR | 64-SR | 81-SR | 100-SR |
|---|---|---|---|---|---|---|---|---|---|
| RAM | 4,2 | 4,8 | 6,3 | 5,5 | 7,5 | 6,2 | 7,2 | 6,7 | 7,3 |
| Swap | 7,8 | 7,9 | 8,4 | 10,5 | 10,9 | 14,7 | 17,4 | 19,8 | 21,5 |

**Tabela 1. Uso de memória (em GBytes): BIRD com OSPF sem falhas.**

sumo de memória oscila, aumentando ou diminuindo conforme o crescimento da grade. Isto ocorre porque a memória RAM é usada em conjunto com a memória virtual. Para

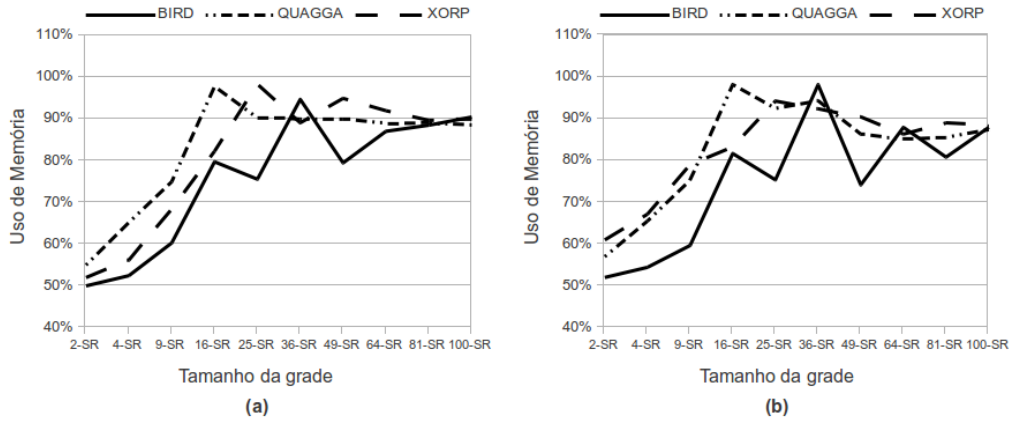


**Figura 6. Uso de memória com OSPF: (a) sem falhas e (b) com falha.**

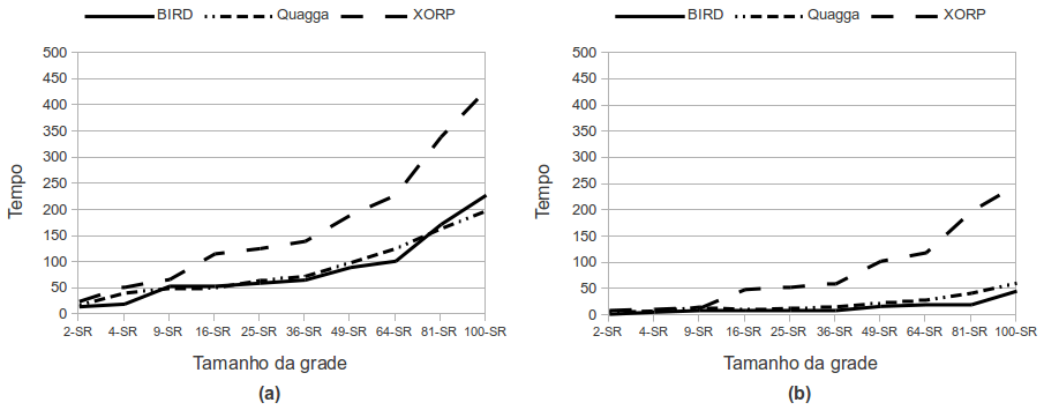


**Figura 7. Tempo de convergência do OSPF: (a) sem falhas e (b) após falha.**

exemplificar é apresentado na Tabela 1 a consumo exigido da RAM e do *swap* nos experimentos em que o BIRD foi configurado com o OSPF nas situações em que não ocorrem falhas. O BIRD foi escolhido porque apresenta maiores oscilações entre as SRs estudadas. Quando a grade aumenta nas dimensões de 4x4, 5x5 e 6x6 existe uma oscilação do consumo da RAM respectivamente de 80,84%, 71,30% e 95,92% do total de memória no entanto, o consumo do *swap* para estas mesmas dimensões é crescente tendo valores de 36,62%, 45,85% e 47,52% respectivamente. A Figura 7a mostra o tempo médio necessário para que o DUT tome conhecimento da topologia da rede quando nenhuma falha ocorre enquanto a Figura 7b mostra o tempo médio necessário após a ocorrência de uma falha. Em média, para convergir a rede após a inicialização dos roteadores, o BIRD precisou de 88,2 segundos, o Quagga precisou de 90,5 segundos e o XORP precisou de 173,3 segundos. Na presença de falha, os tempos de convergência para esses mesmos *softwares* foram, respectivamente, 16,7 segundos, 24,4 segundos e 88,6 segundos. É fácil notar que o XORP possui um pior desempenho quando comparado as outras duas SRs. O BIRD e o Quagga possuem desempenhos próximos, onde a diferença fica em torno de 2,5% para um ou para o outro. Uma possível explicação é que o XORP possui o módulo FEA, específico para gerenciar a RIB e preencher a FIB, enquanto no BIRD e no Quagga as rotas da RIB são transferidas diretamente para a FIB sem qualquer

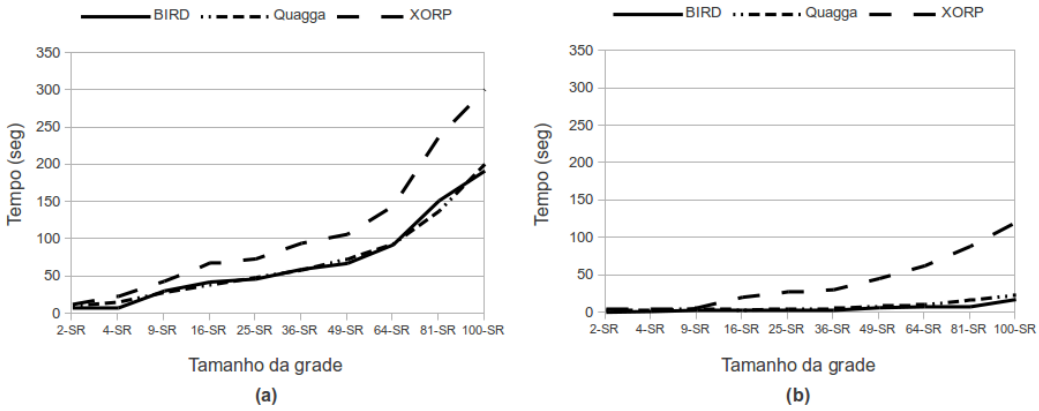


**Figura 8. Tempo rota padrão do OSPF: (a) sem falhas e (b) após falha.**

intermediário (Figura 2).

Em relação ao tempo médio necessário para o DUT descobrir a rota padrão de cada subrede, a Figura 8 indica que o XORP tem o pior desempenho. Os testes mostraram que, em média, o XORP precisa de um tempo 64% maior que o BIRD e o Quagga para descobrir a rota padrão para cada destino após a inicialização e precisa de 20% a mais de tempo para atualizar as rotas após uma falha. Como discutido acima, a provável causa do baixo desempenho do XORP se deve a uma maior hierarquização no preenchimento das tabelas de roteamento da FIB. Mais uma vez, o BIRD e o Quagga apresentam desempenho bem próximos.

### 4.2. BGP

A Figura 9a mostra o uso de CPU quando nenhuma falha ocorre com os roteadores BGP, enquanto a Figura 9b mostra o comportamento das mesmas topologias na presença de falha. O XORP mostrou o pior desempenho em todos os testes realizados (com e sem falha). O BIRD teve o melhor desempenho no ambiente sem falha e no ambiente com falha a partir de 8 ASs. Quando falha ocorre em ambientes com menos de 8 ASs, o XORP é cerca de 3,27% melhor que o BIRD.

O uso da memória nos testes com o BGP é mostrado na Figura 10. Ao comparar esses gráficos com os da Figura 6, é possível observar que o consumo de memória com o BGP é maior. Por exemplo, são necessárias 16 SRs para alcançar 98% de utilização com o OSPF no Quagga, enquanto bastam 4 SRs para atingir 92% de utilização com o BGP no mesmo *software*. Isto pode ser explicado pela forma como o protocolo BGP funciona. No caso em estudo, cada roteador anuncia 1.000 prefixos de redes para todos os outros roteadores. Logo, se a topologia possuir 5 roteadores, serão feitos 5.000. Considerando o uso da memória de acordo com a escalabilidade observa-se que na situação em que um AS possui menos que 3 vizinhos (considerada comum na Internet) o consumo de memória é inferior aos 75% e no caso em que vários vizinhos formam pares (situação observada nos PTTs) a exigência de memória é acima de 90%. Os tempos de convergência e de rota padrão gerados nos testes com o BGP não são aqui apresentados por serem inferiores a 1 segundo. Apenas um roteador atua como intermediário entre os anuncios trocados e desta forma, o tempo necessário para que as informações dos vizinhos alcancem todos os

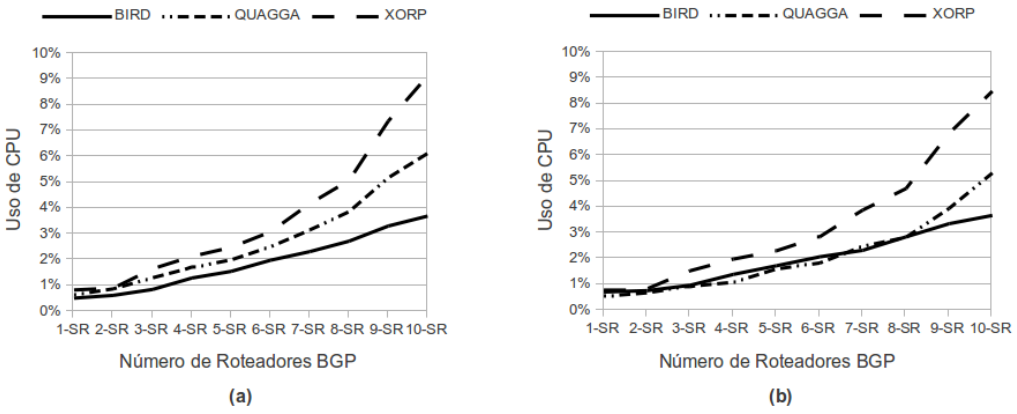


**Figura 9. Uso de CPU com o BGP nos casos (a) sem falhas e (b) com falha**

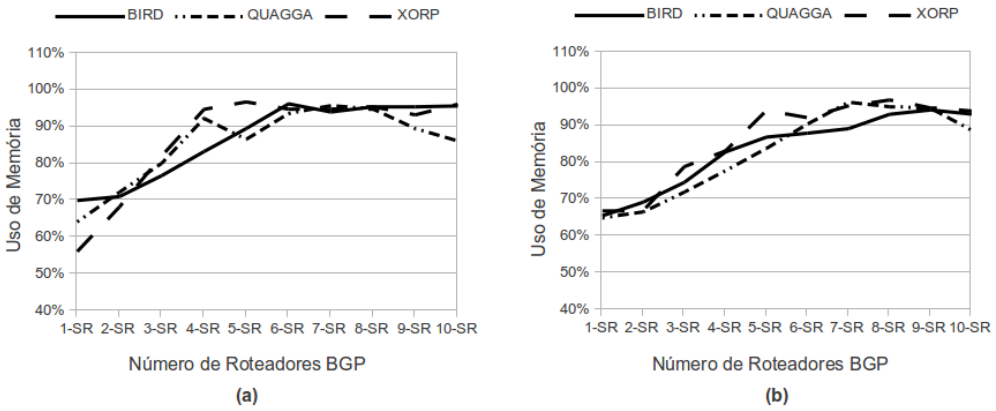


**Figura 10. Uso de Memória com o BGP nos casos (a) sem falhas e (b) com falha**

roteadores é desprezível no ambiente utilizado neste trabalho.

## 5. Conclusões

A utilização mais eficiente dos recursos é um tema de interesse nas pesquisas feitas em SDN. Neste contexto, este trabalho apresenta a comparação das suítes de roteamento BIRD, Quagga e XORP em ambiente virtualizado baseado em contêineres. Foram testadas diferentes topologias e coletadas métricas relacionadas aos tempos, dos roteadores e às cargas de processamento.

A arquitetura da suíte de roteamento do tipo monolítica permitiu ao *kernel* o gerenciamento mais eficiente da CPU e memória. Com relação ao tempo para a criação da tabela de roteamento utilizada pelo sistema operacional, obtiveram os melhores resultados as SRs em que a interligação dos módulos é mais simples possuindo menor número de canais para a troca de dados. Nos experimentos, o BIRD apresentou um desempenho superior na maioria dos testes feitos. O Quagga mostrou um desempenho mais próximo do BIRD, chegando a superá-lo em algumas situações. Por outro lado, o XORP teve normalmente os piores resultados. Atualmente, parece ser o Quagga o preferido entre os desenvolvedores para arquitetura SDN e, devido à facilidade da adição de novos módulos, demandar menos recursos de processamento e apresentar baixa latência no preencimento das tabelas de roteamento usadas pelos sistema operacional.

A escalabilidade da topologia evidenciou a competição pelo *hardware* entre as suítes de roteamento. Este pode ser um dos fatores da perda de desempenho nas situações com maior número de roteadores. Desta forma, pesquisas envolvendo os mecanismos usados pelo sistema operacional para escalonar as suítes de roteamento acrescentarão informações para o dimensionamento do *hardware*.

Como trabalho futuro, se pretende criar cenários que envolvam topologias lógicas independentes usando protocolos de roteamento distintos sobre uma mesma rede e verificar o comportamento da SRs no processo de migração das máquinas virtuais em ambientes tolerantes a falhas.

## Referências


Bhatia, S., Motiwala, M., Muhlbauer, W., Valancius, V., Bavier, A., Feamster, N., Peterson, L., and Rexford, J. (2008). Hosting virtual networks on commodity hardware. In *Techinal Report GT-CS-07-10, January 2008*. Georgia Tech. University.

Bolla, R. and Bruschi, R. (2012). An open-source platform for distributed linux software routers. In *Computer Communications*. Elsevier North-Holland, Inc. `http://dx.doi.org/10.1016/j.comcom.2012.11.00`.

Candelatech. Network testing and emulation solutions. Acessado em Dezembro (2012), `http://www.candelatech.com`.

Carrissimi, A. (2008). Virtualização: da teoria a soluções. In *XXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. SBC.

Corrêa, C., Lucena, S., Rothenberg, C., and Salvador, M. (2011). Uma avaliação experimental de soluções de virtualização para o plano de controle de roteamento de redes virtuais. In *XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. SBC.

Corrêa, C., Lucena, S., Rothenberg, C., and Salvador, M. (2012). Uma plataforma de roteamento como servico baseada em redes definidas por software. In *XVII Workshop de Gerência e Operação de Redes e Serviços*. SBC.

Feamster, N., Balakrishnan, H., Rexford, J., Shaikh, A., and Merwe, J. v. d. (2004). The case for separating routing from routers. In *SIGCOMM'04 Workshops*. ACM.

Hagsand, O., Olsson, R., and Gördén, B. (2008). Towards 10 gb/s open-source routing. Stockholm, Sweden and Uppsala University, Sweden. Royal Institute of Technology.

Handley, M., Hodson, O., and Kohler, E. (2002). Xorp: An open platform for network research. In *First Workshop on Hot Topics in Networks - SIGCOMM*. ACM.

He, L. (2005). Building next generation, open, secure, reliable and scalable experimental ip networks. In *Workshop High Performance Switching and Routing*. IEEE.

Ishiguro, K. (2012). Quagga - a routing software package for tcp/ip networks. GNU Quagga Project. versão 0.99.20mr2.1, `http://www.quagga.org`.

Jasinska, E. and Malayter, C. (2010). (ab)using route servers. In *RIPE60*. Réseaux IP Européens (RIPE) and North American Network Operators Group (NANOG). `http://ripe60.ripe.net`.

Kaliszan, A., Głabowski, M., and Hanczewski, S. (2012). A didactic platform for testing and developing routing protocols. In *The Eighth Advanced International Conference on Telecommunications*. AICT.

Manral, V., White, R., and Shaikh, A. (2004). Benchmarking basic ospf single router control plane convergence. In *RFC 4061*. IETF Network Working Group. `http://tools.ietf.org`.

Nascimento, M. R., Rothenberg, C. E., Salvador, M. R., and Magalhães, M. F. (2011). Routeflow: Roteamento commodity sobre redes programáveis. In *XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. SBC.

Ondrej, F., Mares, M., and Ondrej, Z. (2012). The bird internet routing daemon. CZ.NIC Labs. Acessado em Setembro (2012), `http://bird.network.cz`.

Pica8. Open network. Acessado em Março (2013), `http://www.pica8.com`.

Rothenberg, C., Nascimento, M., Salvador, M., Corrêa, C., Lucena, S., Allan, V., and Verdi, F. L. (2012). Revisiting ip routing control platforms with openflow-based software-defined networks. In *III Workshop de Pesquisa Experimental da Internet do Futuro*. SBC.

Sanghani, B. (2012). 2011 report on european ixps. European Internet Exchange Association. `http://www.euro-ix.net`.

Trosvik, H. (2006). Open source routing suites. In *InteropNet - Open Source Software Initiative, White Paper*. University of Oslo, USIT.