`RDNA Balance`: Balanceamento de Carga por Isolamento de Fluxos Elefante em Data Centers com Roteamento na Origem

Rodolfo V. Valentim[1], Cristina. K. Dominicini[1,2], Diego R. Mafioletti[1,2]
Rodolfo S. Villaca[1], Moises R. N. Ribeiro[1], Magnos Martinello[1]

[1] Núcleo de Estudos em Redes Definidas por Software (Nerds)
Universidade Federal do Espírito Santo (Ufes) – Vitória/ES – Brasil

[2]Instituto Federal do Espírito Santo (Ifes) – Serra/ES, Colatina/ES – Brasil

rodolfo.valentim@aluno.ufes.br, cristina.dominicini@ifes.edu.br
diego.rossi@ifes.edu.br, rodolfo.villaca@ufes.br
moises@ele.ufes.br, magnos@inf.ufes.br

Abstract. *Data center networks, which need to dynamically service a large number of flows with different service requirements, need load balancing mechanisms. However, traditional load-balancing approaches do not allow the full utilization of network resources in a simple, programmable, and scalable way. In this context, this paper proposes* `RDNA Balance` *that exploits elephant flow isolation and source routing, with support in the core of the network, and classsification operations performed on the edge using resources in the OpenFlow protocol. The results show that with this approach it is possible to provide a simple, scalable, and programmable load balancing.*

Resumo. *As redes de Data Center, que precisam atender de forma dinâmica uma grande quantidade de fluxos com diferentes requisitos de serviço, necessitam de mecanismos de balanceamento de carga. Entretanto, as abordagens tradicionais de balanceamento de carga não permitem a completa utilização dos recursos de rede de forma simples, programável e escalável. Nesse contexto, este artigo propõe* `RDNA Balance` *que explora a balanceamento por isolamento de fluxos elefante e roteamento na origem, com suporte no núcleo da rede, e operações de classificação realizadas na borda usando recursos existentes no protocolo OpenFlow. Os resultados mostram que essa abordagem é capaz de prover um balanceamento de carga simples, escalável, e programável.*

1. Introdução

A rápida popularização da computação na nuvem e o surgimento de novas tendências como 5ª Geração de comunicação móvel (5G), Internet das Coisas e Industria 4.0 atribuíram um papel central às redes de *Data Center* (DC), demandando melhoria constante da utilização dos recursos disponíveis nestas redes. A gama de serviços e aplicações que podem ser executados nos DCs é muito grande, variando desde servidores de páginas *web*, mensageiros instantâneos e jogos *online*, até a aplicações que exigem alto poder de computação, tais como análise de dados, aprendizado de máquina e computação científica [Benson et al. 2010].

Essa grande variedade de serviços e aplicações leva aos administradores das redes de *Data Center* o problema de ter de atender diversos fluxos de dados com requisitos

de rede variados e muitas vezes conflitantes [Kandula et al. 2009]. Por exemplo, serviços sensíveis a latência precisam coexistir em uma mesma infraestrutura de rede com serviços que demandam alta vazão de rede e computação. Este cenário exige a existência de esquemas eficientes de engenharia de tráfego que sejam capazes de classificar os fluxos existentes, segregar fluxos com requisitos conflitantes e atribuir caminhos de modo a atender os requisitos sem prejudicar os outros fluxos já existentes.

Entretanto, as redes de *Data Centers* normalmente possuem mecanismos de roteamento complexos e pouco flexíveis, nos quais o encaminhamento é baseado em tabelas. Essa abordagem possui problemas de escalabilidade [Jin et al. 2016] e alto grau de complexidade na gerência dos estados dos fluxos devido à alta transitoriedade dos mesmos. Os mecanismos utilizados pelos balanceadores de carga para realizar o redirecionamento dos fluxos têm que lidar com o roteamento no núcleo da rede. Para topologias de rede muito grandes e um número muito grande de fluxos há o risco do tamanho da tabela de roteamento não ser suficiente para o número de caminhos. Além disso, de modo geral, os mecanismos existentes para definição de rotas precisam instalar regras em todos os *switches* presentes no caminho de um fluxo, o que traz complexidade para aplicação das decisões de balanceamento de carga.

É neste contexto que os mecanismos de roteamento na origem (do inglês *Source Routing* - SR) têm ganhado certa projeção, pois diminuem o tamanho das tabelas de encaminhamento e reduzem a sobrecarga do plano de controle em comparação com as abordagens tradicionais [Martinello et al. 2014]. Mais especificamente, o roteamento rígido na origem (do inglês *Strict Source Routing* - SSR), que permite que, no momento da classificação de um fluxo, uma informação possa ser inserida nos pacotes especificando todo o caminho a ser percorrido na rede. Dessa forma, evita-se a necessidade de consultas a todas as tabelas de encaminhamento nos elementos de comutação existentes no caminho. De acordo com [Jyothi et al. 2015], soluções de roteamento utilizando roteamento na origem demandam uma quantidade menor de regras instaladas e por isso apresentam um grau maior de escalabilidade.

Dentre as diversas alternativas de implementação de SSR em redes de DC destaca-se a RDNA (do inglês *Residue Defined Network Architecture*) [Liberato et al. 2018], que explora características do Sistema Numérico de Resíduos (do inglês *Residue Number System* - RNS) para realizar o roteamento de pacotes no núcleo da rede. A principal característica da RDNA é a divisão entre os elementos de núcleo e borda da rede, onde os elementos de núcleo são mais simples e realizam apenas a tarefa de encaminhamento baseado em operações de módulo, ou seja, sem tabela. Assim, a arquitetura RDNA parece promissora para investigação de novos mecanismos de balanceamento de carga.

Desta forma, este artigo tem como objetivo propor e avaliar a `RDNA Balance`, uma solução para o problema do balanceamento de carga em redes de *Data Center* com rede RDNA. Mais especificamente, deseja-se segregar fluxos longos, que utilizam muita largura de banda (fluxos elefante), dos fluxos de curta duração, que transportam pequenos volumes de dados (fluxos rato). A solução proposta faz uso da RDNA com reescrita de identificadores de rotas nos pacotes usando SSR para alteração de rotas. A `RDNA Balance` é uma abordagem de balanceamento de carga reativa (que reage aos eventos), centralizada (controle em um único lugar, cuja escolha das rotas se apoia na visão centralizada da topologia e da utilização dos enlaces) e localizada no servidor (atua sobre o

servidor, ao invés de *switches*, para realizar o balanceamento).

Foi implementado um protótipo com tecnologias utilizadas em ambientes reais em redes de *Data Centers*, no qual o encaminhamento de pacotes no núcleo da rede é feito sem armazenamento de estados e os *switches* de borda são programáveis utilizando-se de regras *OpenFlow* para fazer a classificação dos fluxos e configuração das rotas. Os resultados obtidos mostram que a `RDNA Balance` permite executar um balanceamento de carga simples, programável e escalável. Simples, pois toda a iteração com o *Data Center* é através do Controlador RDNA, que, por sua vez, precisa instalar regras apenas nos elementos de borda. Programável, porque as bordas da rede usam tabelas *OpenFlow*, que permitem que o balanceador realize diversas ações de configuração através do controlador. E por fim, escalável, em virtude do roteamento não exigir a gerência de estados dos *switches* do núcleo da rede, o que diminui a quantidade de regras por fluxo instaladas na tabela de roteamento [Jyothi et al. 2015].

Por fim, o artigo está organizado da seguinte forma: a Seção 2 traz uma relação de trabalhos relacionados e apresenta as principais contribuições da proposta `RDNA Balance`. A Seção 3 apresenta a proposta da `RDNA Balance`, detalhando a arquitetura, as escolhas de projeto, os mecanismos para atuação para alguns cenários de testes e o protótipo desenvolvido. A Seção 4 avalia a proposta por meio de testes realizados no protótipo. Ao final, a Seção 5 apresenta a conclusão e possíveis trabalhos futuros.

2. Trabalhos Relacionados

Nesta seção serão discutidos alguns trabalhos relacionados ao balanceamento de carga em redes de *Data Center*, em especial quando aplicados em redes com roteamento na origem.

O ECMP [Cai et al. 2012] (do inglês *Equal Cost Multiple Path*) é o método de balanceamento de carga mais comumente utilizado em redes. Utiliza-se de *hashing* para escolher caminhos de mesmo tamanho para os quais deve-se encaminhar os pacotes de um mesmo fluxo. Os pontos fracos deste esquema são bem conhecidos: quando houver colisões na tabela *hash* incorrerá em fluxos utilizando o mesmo caminho e provável desbalanceamento [Al-Fares et al. 2010]. O ECMP também é fraco em topologias de rede assimétricas [Alizadeh et al. 2014].

Presto [He et al. 2015] propõe um esquema de balanceamento de carga na borda da rede. A granularidade do balanceamento está baseada em unidades de fluxo com rajadas de 64KB (*flowcells*). O Presto envia *flowcells* de maneira circular sobre caminhos fim-a-fim pré-configurados. O tamanho é grande o suficiente para não submeter tráfego do tipo rato à reordenação de pacotes e grande o suficiente para quebrar tráfegos elefante em diversas partes. O particionamento dos fluxos em sub-fluxos que percorrem caminhos alternativos leva ao problema da reordenação de pacotes que impõe um gargalo no destino e provável piora na qualidade da experiência dos usuários e aplicações da rede.

Hedera [Al-Fares et al. 2010] defende uma abordagem centralizada para o balanceamento de tráfego. Existe um monitoramento na rede que envia informações dos enlaces e dos fluxos a um controlador. Este controlador é responsável por enviar decisões de roteamento a todos os *switches* do *Data Center* e desta forma realizar o roteamento. É apresentado como uma extensão do ECMP, uma vez no início da operação de um fluxo ele é roteado seguindo o protocolo ECMP, e quando o fluxo atinge um certo limiar é então otimizado.

Tabela 1. Tabela comparativa entre diversos esquemas de Engenharia de Tráfego e Balanceamento de Carga em Redes de *Data Center*

Método	Decisão	Granularidade	Tipo de Ação	Local do Balanceamento	Método de Roteamento
ECMP	Distribuído	Fluxo	Proativo	Rede	Tabela Hash
Presto	Distribuído	Subfluxo	Proativo	Rede	Roteamento na Origem com Tabela (Shadow MACs + OpenFlow)
Hedera	Centralizado	Fluxo	Reativo	Rede	Roteamento com Regras OpenFlow
Planck	Centralizado	Fluxo	Reativo	Rede	Roteamento na Origem com Tabela (Shadow MACs + OpenFlow)
CONGA	Distribuído	Subfluxo	Reativo	Rede	VXLAN
HULA	Distribuído	Subfluxo	Proativo	Rede	P4
RDNA Balance	Centralizado	Fluxo	Reativo	Servidor	Roteamento na Origem sem Tabela

Planck [Rasley et al. 2014], além de ser um mecanismo de engenharia de tráfego, também descreve um método para coleta de dados. Para a coleta de dados utiliza o *port mirroring* de uma ou várias portas para um coletor. Este coletor é responsável por verificar congestionamento e inferir a vazão, disparando eventos de rede ao controlador a cada 4.2 ms – 7.2ms. Uma vez que estes eventos de rede ocorrem o re-roteamento é realizado mudando o *shadow MAC* atribuído a este fluxo que "engana" o *hash* do ECMP e faz com que o fluxo seja roteado para o caminho escolhido.

CONGA [Alizadeh et al. 2014] é um mecanismo de balanceamento de carga em *Data Centers* que identifica globalmente a existência de congestionamento na rede, mas possui atuação distribuída. Utiliza a tecnologia *VXLAN* para o reordenamento dos fluxos com granularidade em nível de *flowlet*.

HULA [Katta et al. 2016] é um método de balanceamento de carga desenvolvido switches programáveis P4 que usa informações de congestionamento global escolher o melhor próximo salto. Desta forma, consegue realizar o balanceamento a nível de *flowlets* de maneira proativa.

Todas essas soluções diferem bastante entre si mas sofrem do mesmo problema: não atendem os requisitos de escalabilidade, pois requisitam grande esforço na gerência dos estados nos *switches* existentes no caminho de cada fluxo. Esse esforço, somado a alta taxa de instalação de novos fluxos resulta em um número muito grande de estados para se gerenciar no núcleo e borda da rede do *Data Center*.

A Tabela 1 resume os trabalhos citados anteriormente. Avalia-se na tabela: i) o local da decisão de balanceamento (centralizado ou distribuído); ii) o tamanho do fluxo redirecionado (pacote, fluxo ou subfluxo); iii) se o método reage a congestionamento ou evita antes de acontecer (proativo ou reativo); iv) se o balanceador atua nos elementos de rede ou nos servidores (rede ou servidor) e; v) o método de seleção das rotas (Método de Roteamento). Esta última característica varia muito entre diferentes abordagens mas importante de se destacar, pois é a principal mudança da abordagem proposta (`RDNA Balance`) para as demais propostas existentes na literatura.

A partir dos resultados comparativos com o estado da arte das soluções de Engenharia de Tráfego e Balanceamento de Carga em redes de *Data Center* (Tabela 1), as principais contribuições deste trabalho estão resumidas conforme segue:

1. Implementação de um mecanismo programável e eficiente para segregação de flu-

xos elefantes e ratos em uma arquitetura de *Data Center* com roteamento na origem e separação entre borda e núcleo.
2. Apresentação da viabilidade da proposta, expondo que o roteamento na origem possibilita a migração entre caminhos disponíveis nas redes de *Data Center* com baixa taxa de perda de pacotes, possibilitando a migração de fluxos sem afetar a qualidade da experiência dos usuários.
3. Instanciação de um protótipo utilizando a arquitetura RDNA em uma topologia de rede realística. As análises de desempenho dos mecanismos e a viabilidade da implementação correspondem ao que seria encontrado em um ambiente de *Data Center* real.
4. Extensão da RDNA para prover um mecanismo ágil de balanceamento de carga, permitindo a realização de ações de Engenharia de Tráfego na rede.

3. `RDNA Balance`

Uma característica do tráfego existente nas redes de *Data Center* modernos é a intensa comunicação entre os servidores da infraestrutura. Sobre os fluxos provenientes dos servidores, [Kandula et al. 2009, Benson et al. 2010] afirmam que a cada ms 100 novos fluxos chegam a um *Data Center*; fluxos de até $25s$ são responsáveis por mais da metade do volume de tráfego; e apenas 0.1% do total de fluxos duram mais de $100s$, sendo responsáveis por quase 20% do volume de dados. Outra observação importante é que uma pequena fração dos enlaces experimenta perdas muito maiores do que o restante da rede. Demonstra-se, assim, que é possível utilizar caminhos alternativos para evitar perdas [Benson et al. 2010] e melhorar a qualidade do serviço provido para os fluxos de rede no *Data Center*.

Esta característica da natureza dos fluxos em um *Data Center* necessita de um mecanismo de balanceamento de fluxo que seja escalável, capaz de lidar com a gigante quantidade de fluxos ingressantes e, além disso, seja capaz de melhorar a utilização dos recursos de rede, segregando os fluxos elefante dos outros fluxos com requisitos diferenciados. A segregação deve ocorrer de forma dinâmica, válida somente durante o tempo de vida do fluxo. A migração de rotas no *Data Center* deve ocorrer de maneira transparente, causando o mínimo de interferência possível à qualidade da experiência dos usuários das aplicações.

Nossa proposta é definida pela `RDNA Balance`, uma solução de balanceamento que atua na arquitetura de rede RDNA realizando a separação entre os fluxos elefantes e os outros fluxos. Essa separação visa melhorar a utilização da largura de banda disponível entre os diversos caminhos do *Data Center* e, ao mesmo tempo, garantir a menor latência possível para os diversos fluxos concorrentes por meio de uma solução de Balanceamento de Carga e Engenharia de Tráfego na rede.

A arquitetura de rede RDNA argumenta que para novos *Data Centers* serem capazes de atender a novas demandas de baixa latência e, além disso, ser programável, é preciso fazer uma clara distinção entre o hardware que executa o encaminhamento no plano de dados e o software que controla a rede.

Seguindo os princípios das Redes Definidas por Software, (do inglês *Software Defined Networking* - SDN), a RDNA é composta por três elementos representados na Figura 1: i) o controlador RDNA, que consiste em um controlador centralizado capaz

de configurar políticas e monitorar os *switches* da rede; ii) *switches* de borda, que são os responsáveis por inserir nos pacotes os identificadores das rotas usando a codificação possibilitada pelo Teorema Chinês dos Restos [Liberato et al. 2018] e; iii) os *switches* de núcleo, que não possuem tabelas e realizam o encaminhamento dos pacotes realizando operações de módulo. A operação de módulo é realizada entre o ID da rota presente no cabeçalho do pacote (inserido no pacote pelo *switch* de borda) e o identificador do *switch* de núcleo. O resultado indica a porta de saída para a qual o pacote deve ser encaminhado. Na Figura 1, os números presentes nos *switches* de núcleo são seus IDs e são todos coprimos entre si. Por exemplo, um pacote cujo MAC de Destino foi alterado na borda para o endereço de rota ($R = 32338$), ao chegar no *Switch* 37 a operação de módulo $< 32338 >_{37}$ determina que a porta de saída deste pacote é a porta 0 (resto da divisão é igual a 0).

Na proposta da arquitetura RDNA ressalta-se a possibilidade de se implementar mecanismos de balanceamento de carga, mas não havia, até então, o desenvolvimento dessa ideia. A solução de balanceamento de carga `RDNA Balance` é apresentada na Figura 1, juntamente com a sua interação com a arquitetura RDNA. Propõe-se que toda a iteração do balanceador de carga com a infraestrutura se dê através do controlador, o que simplifica o funcionamento do balanceador, que agora possui uma única interface de configuração. Dentro do bloco representando a `RDNA Balance` encontram-se dispostos 4 blocos funcionais e o banco de dados. Tais blocos representam as principais funcionalidades necessárias para realização de um balanceamento de carga efetivo pela RDNA Balance.

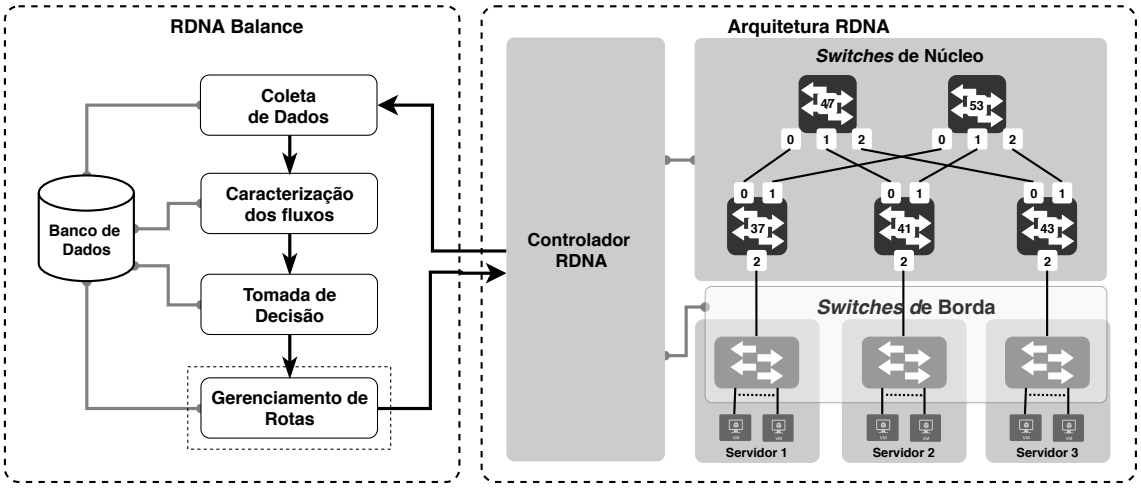

Figura 1. Solução RDNA Balance para balanceamento de carga na arquitetura RDNA.

As principais funcionalidades previstas na solução `RDNA Balance` são: i) monitorar a rede para coletar informações sobre topologia e uso de recursos; ii) caracterizar os fluxos de acordo com a utilização de recursos; iii) tomar decisão de balanceamento de modo a melhorar a utilização de recursos; iv) gerenciar rotas atuando sobre os elementos de rede, implementando na rede as decisões de balanceamento. Neste trabalho, focaremos nos mecanismos de atuação sobre os elementos de rede de modo a obter um balanceamento de carga reativo, centralizado e localizado no servidor (*funcionalidade iv*).

A seguir serão apresentadas as escolhas de projeto que validam as funcionalidades

de coleta, caracterização e tomada de decisão. Em seguida apresentaremos detalhes sobre a solução de gerenciamento de rotas.

3.1. Escolhas de projeto

- **Programabilidade nas Bordas**: Um dos pontos fortes da proposta é a programabilidade das bordas. Ao se utilizar *switches* OpenFlow na borda da rede, tem-se a disposição: i) possibilidade de realizar a classificação dos fluxos com base em informações que vão desde a camada 1 até a camada 4 da pilha TCP/IP; ii) possibilidade de se modificar o cabeçalho dos pacotes de fluxos de rede e; iii) diversas ações de encaminhamento, que vão desde o redirecionamento de dados para uma porta específica, encaminhamento dos pacotes para o controlador da rede, ou descarte de pacotes. Além disso, o uso do OpenFlow nas bordas habilita o monitoramento do tráfego nos *switches* através de contadores específicos para cada regra de fluxo. Com estes contadores, torna-se possível caracterizar os fluxos e propiciar tomada de decisões de encaminhamento.
- **Controle Centralizado**: Abordagens distribuídas de balanceamento de carga muitas vezes tomam decisões baseadas em informações locais sobre o uso dos enlaces da rede [Cai et al. 2012] e, por isso, não são otimizadas [Alizadeh et al. 2014]. Abordagens centralizadas de balanceamento de carga já foram demonstradas em [Alizadeh et al. 2014, Rasley et al. 2014] como sendo melhores opções para se otimizar o uso da rede. Na arquitetura RDNA, o controlador se comunica com os *switches* de borda e núcleo. Inicialmente, o controlador é responsável: i) pela descoberta de topologia por meio da análise de pacotes LLDP (*Link Layer Discovery Protocol*); ii) pela configuração inicial dos *switches* e; iii) pela instalação proativa das regras de fluxos conhecidos na borda da rede. A RDNA se aproveita desta visão centralizada para coletar informações dos fluxos instalados a fim de caracterizá-los e, em conjunto com as informações topológicas, tomar decisões que permitam melhorar a utilização dos recursos da rede. Há estudos que propõem controladores centralizados resilientes a falha que mantém o estado dos switches utilizando um esquema de mestre-escravo [Katta et al. 2015].

3.2. Gerenciamento de Rotas e Mecanismos de Atuação

Para demonstrar os mecanismos de atuação que habilitam o gerenciamento de rotas na `RDNA Balance`, foram elaborados diferentes cenários para Engenharia de Tráfego. A partir desses cenários discute-se as ações necessárias para se obter melhor aproveitamento dos recursos de rede disponíveis no *Data Center* gerenciado pela `RDNA Balance`. Os cenários de interesse escolhidos são: i) fluxo elefante concorrendo com fluxo rato; ii) fluxo elefante concorrendo com outro fluxo elefante; iii) fluxo com requisitos rígidos de confiabilidade, ou seja, baixa tolerância a perda de pacotes. Nestes cenários são necessários os seguintes mecanismos de atuação: definir rota de um fluxo ingressante, migrar a rota de um fluxo e multiplicar (duplicar) um fluxo ingressante.

- **Definir a Rota de um Fluxo Ingressante**: o primeiro pacote de um fluxo ingressante desconhecido não possui nenhuma regra associada ao chegar a um *switch* de borda, e por isso é enviado para o controlador. O controlador precisa alocar uma rota específica a este fluxo e instalar, nas bordas da rede, uma regra que insira o identificador de rota associada. Esse identificador é inserido no *switch* de ingresso

e removido no *switch* egresso. A regra na origem modifica o cabeçalho do pacote, reescrevendo o campo MAC de destino com o identificador da rota a ser usada. A regra no destino restaura o MAC de destino original dos pacotes.

- **Migrar a Rota de um Fluxo**: No total, para cada fluxo existente são necessárias apenas 2 regras para se estabelecer o roteamento (bordas de ingresso e egresso). O `RDNA Balance` realiza a modificação de apenas duas regras para migrar de uma rota para a outra. Esta é uma vantagem quando comparada com outras propostas, como Hedera e CONGA, que precisam atualizar todas as tabelas de roteamento dos *switches* no caminho entre origem em destino. Na `RDNA Balance` a migração de caminhos é realizada modificando-se somente as duas regras instaladas no momento da instanciação do fluxo na borda da rede.
- **Multiplicar um Fluxo**: com o uso do protocolo OpenFlow é possível criar grupos de regras que permitem realizar ações a determinados fluxos. O grupo do tipo `ALL` intercepta qualquer pacote recebido como entrada e o multiplica, de forma que os pacotes possam ser operados de forma independente por cada ação do grupo. Aplicada à realidade do balanceamento de carga, com esta solução é possível multiplicar o envio dos pacotes por múltiplas rotas visando aumentar a confiabilidade na entrega mesmo em situações de falha nos enlaces da rede. É uma solução que pode ser usada em aplicações que necessitam de maior confiabilidade na entrega dos dados.

Em uma solução de Balanceamento de Carga em redes de *Data Center* é importante que haja o mínimo de perda nos dados durante a migração dos fluxos para novos caminhos e, para isso, na `RDNA Balance` segue-se a seguinte ordem: i) inicialmente instala-se a regra no destino, sem remover a regra antiga; ii) modifica-se a regra na origem ou instala-se uma regra na origem com uma prioridade maior; iii) uma vez que a nova rota esta configurada, as regras antigas no destino são removidas.

Conforme já mencionado, o roteamento utilizando tabelas no núcleo da rede não escala. *Data Centers* recebem um grande número de fluxos novos a cada instante, o que implica em uma enorme quantidade de regras a serem instaladas e estados a serem gerenciados em cada *switch* e roteador entre a origem e o destino de um fluxo. Este é um dos problemas tratados na abordagem do `RDNA Balance`, pois só há necessidade de gerenciar os estados dos *switches* de borda, com menos elementos a serem gerenciados.

A solução proposta, `RDNA Balance`, é capaz de instalar as regras na borda da rede com baixíssimo tempo de propagação, da ordem de 1 ms por regra, sem perdas significativas no fluxo migrado. Este custo é muito baixo dado o alto grau de programabilidade das redes RDNA e a versatilidade da solução `RDNA Balance`. Estes resultados serão apresentados na Seção 4. Análises de escalabilidade do encaminhamento e mais detalhes do roteamento são encontrados em [Liberato et al. 2018].

3.3. Implementação do Protótipo da `RDNA Balance`

Nesta seção será descrita a implementação do protótipo da solução `RDNA Balance` utilizando a arquitetura RDNA. Implementou-se uma rede *two-tier*, porém com os *switches* sendo virtualizados em servidores. Cada *switch* de núcleo está virtualizado em um servidor físico dedicado e a conexão entre os servidores é realizada por meio de suas interfaces de rede. A virtualização dos *switches* permite maior facilidade em implementar o rotea-

mento utilizando operações de módulo. Tanto os *switches* de núcleo quando os *switches* de borda são implementados como *bridges* no Open vSwitch (OvS).

A *framework* de nuvem que faz a virtualização dos recursos dos servidores é uma versão modificada do *OpenStack*. Além disso, o *datapath* do OvS foi modificado para dar suporte ao encaminhamento por módulo. O protótipo é composto por 9 servidores físicos com Linux Ubuntu, conforme pode ser visto na Figura 2. Cada servidor tem um processador Intel Xeon E5-2620 de 2,4 GHz, 16 GB de memória e quatro ou cinco NICs Ethernet de 1 Gbps: uma conectada à rede de dados *OpenStack*, uma conectada à rede de gerenciamento *OpenStack* e as interfaces restantes são conectadas aos outros servidores para construir essa topologia.

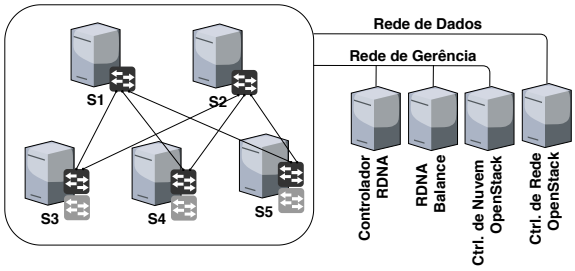

Figura 2. Representação do protótipo implementado (Prova de Conceito).

Compõem a infraestrutura: 5 nós de computação (S1-S5), 1 nó Controlador da Rede *OpenStack*, 1 nó Controlador RDNA, 1 nó de Controlador da Nuvem *OpenStack* e 1 nó com a solução de balanceamento de carga `RDNA Balance`. O Controlador RDNA foi implementado utilizando o *framework* Ryu. O `RDNA Balance` foi implementado em *Python*. A comunicação entre o Controlador RDNA e o `RDNA Balance` é realizada por meio de uma API REST.

4. Avaliação da `RDNA Balance`

Foram elaborados cenários de teste que seguem os casos apresentados durante a elaboração da proposta. São eles: i) fluxo elefante concorrendo com fluxo rato; ii) fluxo elefante concorrendo com outro fluxo elefante; iii) fluxo com requisitos estritos de confiabilidade na entrega.

Inicialmente, avaliou-se a viabilidade da aplicação de mecanismos de engenharia de tráfego nos três cenários, destacando-se a necessidade da ação direta de um balanceador de carga. Nestes cenários, verifica-se que após a ação da `RDNA Balance`, segregando fluxos rato e elefante, há um ganho na vazão de fluxos elefante e uma diminuição da latência dos fluxos rato. O objetivo desse conjunto de testes é a realização de uma análise qualitativa do protótipo, demonstrando seu funcionamento e viabilidade de implementação em um protótipo real.

Em seguida realizou-se uma avaliação de desempenho do custo da migração de rotas por meio da solução `RDNA Balance`. Partiu-se da premissa de que não é viável que uma troca de caminho para um fluxo incorra em grande perda de pacotes durante a migração, ou seja, que a migração demore a ser efetivada. Mostrou-se que os mecanismos escolhidos apresentam uma perda na média de 0.5% de pacotes durante o processo de migração, que dura cerca de $1ms$.

4.1. Cenários de Engenharia de Tráfego

Nesta seção exploramos os cenários que motivaram a proposição da solução `RDNA Balance` e seus mecanismos de atuação. Os experimentos realizados nesta seção mostram o funcionamento da solução e a viabilidade de sua implementação em uma rede RDNA implementada com artefatos reais em ambiente de produção.

4.1.1. Fluxo Elefante concorrendo com Fluxo Rato

Considere o cenário proposto na Figura 3(a). Neste cenário dois fluxos compartilham um mesmo enlace: 1 fluxo UDP de $930Mbps$, caracterizado como um fluxo elefante e 1 mensagem ICMP to tipo *ping*, enviada a cada um segundo, caracterizada como fluxo rato. O fluxo UDP é gerado pela ferramenta *pktgen* que é capaz de gerar pacotes de tamanhos específicos a uma taxa constante. O *pktgen* ao gerar a taxa solicitada no experimento de $81274pps$ (pacotes por segundo) de $1518Bytes$ saturou o limite físico do enlace de $930Mbps$. A taxa e o tamanho do pacote estão de acordo com [Bolla and Bruschi 2006, Popoviciu et al. 2008]. Antes da migração, a rota do fluxo UDP é $VMS1 \rightarrow S_{11} \rightarrow S_{19} \rightarrow S_{17} \rightarrow VMD1$ e do fluxo ICMP é $VMS2 \rightarrow S_{11} \rightarrow S_{19} \rightarrow S_{17} \rightarrow VMD2$.

No instante de tempo igual a $30s$, força-se a atuação da solução `RDNA Balance` com a migração da rota do fluxo UDP para a rota $VMS1 \rightarrow S_{11} \rightarrow S_{13} \rightarrow S_{17} \rightarrow VMD1$. Após a migração os fluxos são separados e não há compartilhamento de enlaces físicos entre os dois fluxos, como pode ser visto na Figura 3(b). Neste cenário, o gargalo para a latência é o enlace físico, pois o barramento virtual conectando o *switch* de núcleo com o de borda possui capacidade de vazão muito maior. A medida de latência é obtida pela ferramenta *ping*. Durante o experimento, amostrou-se a latência percebida pelo fluxo rato a cada $1s$ e o resultado é mostrado na Figura 3(c). Percebe-se uma queda brusca de $13ms$ para $0.7ms$ na latência do tráfego ICMP percebida em $VMD2$ no instante $30s$, decorrente da migração do fluxo elefante para outra rota na rede RDNA.

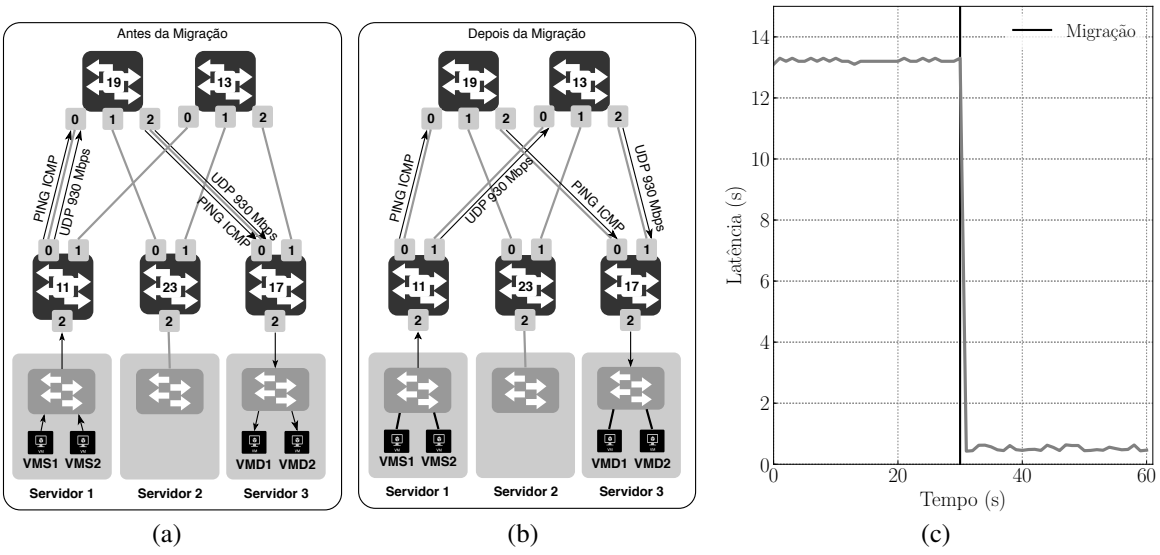

Figura 3. Migração de um fluxo e a análise do impacto sobre a latência percebida em em $VMD2$ durante o experimento. (a) Rotas antes da migração (b) Rotas depois da migração (c) Latência percebida em $VMD2$ durante o experimento.

4.1.2. Fluxo Elefante concorrendo com outro Fluxo Elefante

O cenário de testes da Figura 4(a) ilustra dois fluxos elefante com comportamentos distintos compartilhando o mesmo enlace $S_{19} \rightarrow S_{17}$. Neste cenário, o fluxo TCP, adaptativo, ($VMS1 \rightarrow S_{11} \rightarrow S_{19} \rightarrow S_{17} \rightarrow VMD1$) é prejudicado por compartilhar o mesmo enlace com o fluxo UDP ($VMS2 \rightarrow S_{23} \rightarrow S_{19} \rightarrow S_{17} \rightarrow VMD2$). O fluxo UDP ocupa $300 Mbps$ da banda do enlace e fluxo TCP ocupa aproximadamente os 630 $Mbps$. A taxa atingida pelo fluxo TCP se deve ao limite da interface física de 1 $Gbps$ do servidor e a taxa do fluxo UDP simula um fluxo elefante. Aos 30 s, força-se a atuação da solução `RDNA Balance` e migra-se o fluxo TCP, isolando-o de seu concorrente.

A Figura 4(c) mostra o ganho de vazão percebida em $VMD1$ no instante 30 s resultante da migração. Destaca-se que a rapidez na migração da rota no fluxo TCP não afeta significativamente a vazão percebida pelo protocolo, que possui características intrínsecas de controle de fluxo. A ausência de perda de pacotes e retransmissão faz com que a vazão fique estável, comprovando a rapidez do mecanismo de troca que demonstra um balanceamento de carga simples e eficiente.

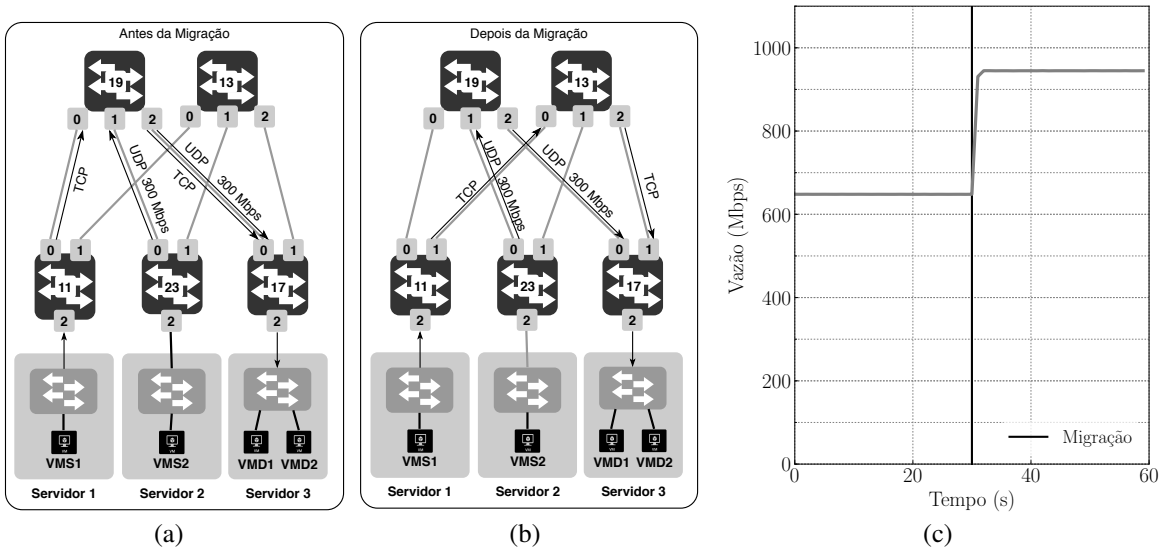

Figura 4. Migração de um fluxo TCP e a análise do impacto sobre a banda. (a) Rotas antes da migração (b) Rotas depois da migração (c) Banda percebida em $VMD1$ (tráfego TCP) durante o experimento.

4.1.3. Fluxo com requisitos de confiabilidade na entrega

Como pode ser visto na Figura 5(a), o grupo de regras do tipo *ALL* foi usado neste cenário para enviar pacotes de um fluxo de 300 $Mbps$ com requisitos estritos de entrega. No experimento optou-se por duplicar o fluxo por duas rotas distintas: $VMS1 \rightarrow S_{11} \rightarrow S_{13} \rightarrow S_{17} \rightarrow VMD1$ e $VMS1 \rightarrow S_{11} \rightarrow S_{19} \rightarrow S_{23} \rightarrow S_{13} \rightarrow S_{17} \rightarrow VMD1$. No instante de 30 s, simula-se a queda de uma interface física, marcada com um X na Figura 5(b), impossibilitando a chegada de um dos fluxos ao destino. Como adotou-se redundância de caminhos, a vazão percebida em $VMD1$, que antes era de 600 $Mbps$, passou a receber 300 $Mbps$. Esse fluxo foi gerado com a ferramenta *iperf*.

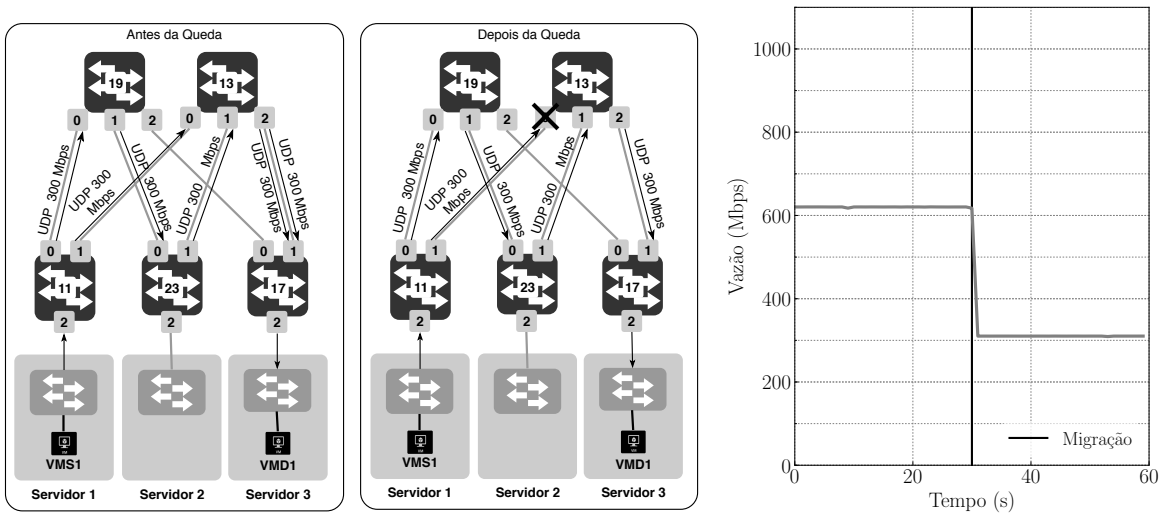

Figura 5. Queda de uma interface física e a análise do impacto sobre a banda em um cenário de confiabilidade na entrega. (a) Rotas antes da queda (b) Rotas depois da queda (c) Banda percebida em $VMD1$ durante o experimento.

4.2. Avaliação do Custo da Migração

O custo da migração é a perda de pacotes durante a migração de rotas devido ao tempo necessário para se completar todo o processo de configuração da rede, tanto na borda quanto no núcleo, para realizar o direcionamento do tráfego em todos os *switches* no caminho. Quanto maior o tempo de migração e a banda transmitida, maior é a perda de pacotes de um determinado fluxo, e isso impacta diretamente na diminuição da vazão percebida pelos fluxos que sofreram migração. Destaca-se que nas redes que realizam o roteamento por consulta em tabelas, usadas por soluções como o Hedera e CONGA, uma migração envolve atualização dos estados de todas tabelas no caminho entre origem e destino. Em uma rede RDNA o mecanismo da `RDNA Balance` só precisa atuar nos equipamento de borda, ingresso e egresso, independente do comprimento do caminho a ser percorrido no núcleo da rede.

Neste experimento, criou-se um fluxo elefante UDP com diferentes requisitos de largura de banda, variando de 100 $Mbps$ até 800 $Mbps$, duplicando-se esse requisito a cada iteração. A rota antes da migração é $VMS1 \rightarrow S_{11} \rightarrow S_{19} \rightarrow S_{17} \rightarrow VMD1$. Após 50 s, realizou-se a migração para a rota $VMS1 \rightarrow S_{11} \rightarrow S_{13} \rightarrow S_{17} \rightarrow VMD1$ de mesmo comprimento. Na Figura 6(c) encontra-se a medição da vazão recebida em $VMD1$ para cada largura de banda avaliada. As medições de vazão recebida foram feitas utilizando a ferramenta *bwm-ng* com tempo de amostragem de 1 s e o tráfego foi gerado utilizando a ferramenta *iperf*.

Verifica-se uma pequena perda de pacotes, resultando em uma pequena queda na banda percebida no destino do fluxo. Esta perda acontece durante o breve período em que uma regra é alterada na origem. Já os pacotes em trânsito durante a migração, que possuem o identificador da rota antiga no cabeçalho *Ethernet*, são capazes de chegar ao destino sem serem perdidos. A perda é da ordem de apenas 0.5% da vazão total do fluxo, o que comprova que mesmo em situações de alta vazão o tempo de modificação de uma regra OpenFlow é da ordem de poucos ms.

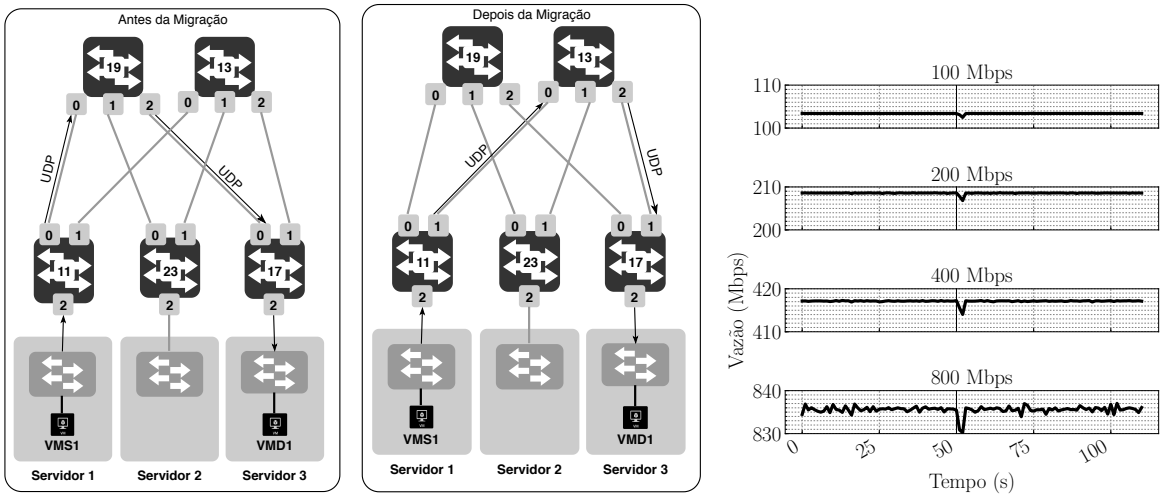

Figura 6. Migração de um fluxo e a análise do impacto sobre a banda. (a) Rotas antes da migração (b) Rotas depois da migração (c) Banda percebida em $VMD1$ durante o experimento.

5. Conclusões e Trabalhos Futuros

A solução `RDNA Balance` possui um grau de programabilidade muito interessante para aplicações de Balanceamento de Carga. A avaliação da prova de conceito implementada com tecnologias existentes em ambientes de produção de *Data Centers* reais demonstraram simplicidade, eficiência e programabilidade. A solução apresentou-se como um mecanismo simples e de ação rápida com ganhos muito claros em sua implementação.

Foram apresentados cenários onde o uso de uma abordagem centralizada, reativa e estritamente roteada na origem traz vantagens sobre outras abordagens distribuídas e com mecanismos de roteamento mais tradicionais. Os resultados mostraram a rapidez na implementação de novos caminhos, com baixa taxa de perdas, sem comprometimento no funcionamento dos serviços associados aos fluxos que tiveram seus caminhos modificados. Além disso, demonstrou-se flexibilidade na escolha de possíveis caminhos e explorou-se a possibilidade de redundância de caminhos para fluxos com requisitos de alta confiabilidade na entrega.

Para trabalhos futuros existem lacunas a serem preenchidas na detecção de congestionamento, caracterização dos fluxos, cenários mais realistas e métricas de aplicação. Alguns estudos sugerem realizar coleta de informações dos fluxos por amostragem e um estudo mais aprofundado sobre o assunto pode trazer ganhos importantes para a escalabilidade da solução. Além disso, planeja-se estender a análise de desempenho da `RDNA Balance` e implementar uma solução concorrente baseada em tabelas para comparação.

6. Agradecimentos

Este trabalho recebeu financiamento parcial do projeto Horizon 2020 (União Européia) sob no. 688941 (FUTEBOL), assim como do MCTI por meio da RNP e do CTIC, e financiamento parcial proveniente de recursos de bolsas e projetos CNPq e FAPES. Além disso, recebeu apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Referências

Al-Fares, M. et al. (2010). Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX NSDI*, NSDI'10, pages 19–19, CA, USA. USENIX.

Alizadeh, M. et al. (2014). CONGA: Distributed Congestion-Aware Load Balancing for Datacenters. *ACM SIGCOMM Computer Communication Review*, 44(4):503–514.

Benson, T. et al. (2010). Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40(1):92–99.

Bolla, R. and Bruschi, R. (2006). Rfc 2544 performance evaluation and internal measurements for a linux based open router. In *High Performance Switching and Routing, 2006 Workshop on*, pages 6–pp. IEEE.

Cai, Y. et al. (2012). RFC 6754 - protocol independent multicast equal-cost multipath (ECMP) redirect. Technical report.

He, K. et al. (2015). Presto: Edge-based Load Balancing for Fast DCN. *Sigcomm 2015*, pages 465–478.

Jin, X. et al. (2016). Your data center switch is trying too hard. In *Proceedings of the Symposium on SDN Research*, SOSR '16, pages 12:1–12:6, NY, USA. ACM.

Jyothi, S. A. et al. (2015). Towards a flexible data center fabric with source routing. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 10. ACM.

Kandula, S. et al. (2009). The nature of data center traffic. In *Proceedings of the 9th ACM SIGCOMM conference on IMC '09*, page 202, New York, New York, USA. ACM.

Katta, N. et al. (2015). Ravana: Controller fault-tolerance in software-defined networking. In *Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research*, page 4. ACM.

Katta, N. et al. (2016). Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*, SOSR '16, pages 10:1–10:12, NY, USA. ACM.

Liberato, A. et al. (2018). RDNA: Residue-Defined Networking Architecture Enabling Ultra-Reliable Low-Latency Datacenters. *IEEE TNSM*, 4537(c):1–1.

Martinello, M. et al. (2014). Keyflow: A prototype for evolving SDN toward core network fabrics. *Network, IEEE*, 28:12–19.

Popoviciu, C., Hamza, A., Van de Velde, G., and Dugatkin, D. (2008). RFC 5180 - IPv6 benchmarking methodology for network interconnect devices. Technical report.

Rasley, J. et al. (2014). Planck: Millisecond-scale Monitoring and Control for Commodity Networks Jeff. In *Proceedings of the 2014 ACM conference on SIGCOMM - SIGCOMM '14*, pages 407–418, New York, New York, USA. ACM Press.