UNIVERSIDADE DO ESTADO DE SANTA CATARINA - UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS - CCT
MESTRADO EM COMPUTAÇÃO APLICADA

VILSON MORO

CARACTERIZAÇÃO DO IMPACTO DA VIRTUALIZAÇÃO DO CONTROLE DE CONGESTIONAMENTO NA EXECUÇÃO DO HADOOP MAPREDUCE

JOINVILLE

2018

VILSON MORO

CARACTERIZAÇÃO DO IMPACTO DA VIRTUALIZAÇÃO DO CONTROLE DE CONGESTIONAMENTO NA EXECUÇÃO DO HADOOP MAPREDUCE

Dissertação submetida ao Programa de Pós-Graduação em Computação Aplicada do Centro de Ciências Tecnológicas da Universidade do Estado de Santa Catarina, para a obtenção do grau de Mestre em Computação Aplicada.

Orientador: Dr. Guilherme Piêgas Koslovski

JOINVILLE

2018

Vilson Moro
Caracterização do Impacto da Virtualização do Controle de Congestionamento na Execução do Hadoop MapReduce/ Vilson Moro. – Joinville, 2018-
89 p. : il. (algumas color.) ; 30 cm.

Dr. Guilherme Piêgas Koslovski

– Universidade do Estado de Santa Catarina - UDESC, 2018.

1. Tópico 01. 2. Tópico 02. I. Prof. Dr. xxxxx. II. Universidade do Estado de Santa Catarina. III. Centro de Ciências Tecnológicas. IV. identificação xxxx

CDU 02:121:005.7

Caracterização do Impacto da Virtualização do Controle de Congestionamento na Execução do Hadoop MapReduce

por

Vilson Moro

Esta dissertação foi julgada adequada para obtenção do título de

Mestre em Computação Aplicada

Área de concentração em "Ciência da Computação",
e aprovada em sua forma final pelo

CURSO DE MESTRADO ACADÊMICO EM COMPUTAÇÃO APLICADA
DO CENTRO DE CIÊNCIAS TECNOLÓGICAS DA
UNIVERSIDADE DO ESTADO DE SANTA CATARINA.

Banca Examinadora:

Prof. Dr. Guilherme Piêgas Koslovski
CCT/UDESC (Orientador/Presidente)

Prof. Dr. Maurício Aronne Pillon
CCT/UDESC

Prof. Dr. Tiago Antonio Rizzetti
CTISM/UFSM

Joinville,SC, 13 de dezembro de 2018.

Dedico este trabalho aos meus familiares, amigos, colegas e professores que me acompanharam e me deram forças nessa magnífica trajetória.

AGRADECIMENTOS

Desejo agradecer, primeiramente a Deus por me conceder saúde e serenidade para realizar este trabalho; Minha esposa Maricarla pelo incentivo e confiança; Minhas filhas: Bárbara, Beatriz e Bruna que são a razão da minha vida; ao orientador pelo comprometimento e presteza no acompanhamento das atividades e aos demais professores e colegas que contribuíram ouvindo e sugerindo ações a serem tomadas para a conclusão deste trabalho.

"Independentemente das circunstâncias, devemos ser sempre humildes, recatados e despidos de orgulho."
Dalai Lama

RESUMO

As dificuldades gerenciais e financeiras enfrentadas pelas empresas, quanto a manutenção de um parque tecnológico, contribuíram para o surgimento dos provedores de Infraestruturas como Serviço (IaaS) que oferecem serviços virtualizados de processamento, armazenamento e comunicação. Assim, surgem os *data center*s (DCs) de nuvens computacionais que realizam as tarefas voltadas ao gerenciamento de recursos, enquanto os inquilinos se concentram nas atividade de negócios. Provedores de nuvens hospedam máquinas virtuais (MVs) de múltiplos inquilinos, configuradas com versões distintas de sistema operacionals (SOs), bibliotecas e algoritmos para o *Transmission Control Protocol* (TCP) otimizados e não otimizados. A variedade de algoritmos e configurações relacionados com o protocolo TCP constitui um cenário de comunicação heterogêneo. Algoritmos otimizados de controle de congestionamento tendem a serem mais agressivos na competição pelo compartilhamento de largura de banda e se recuperam de perdas causadas por congestionamento de forma mais eficaz. Assim, as aplicações que usam um algoritmo não otimizado tem um cenário hostil que pode inviabilizar a comunicação. Essa situação de falta de equidade nos fluxos, entre diferentes inquilinos afeta o sentido de justiça no uso da rede, indicando para os clientes uma visão equivocada sobre a qualidade do serviço oferecido pelo provedor. Devido ao total controle gerencial no DC, provedores podem aplicar a virtualização do controle de congestionamento para traduzir os algoritmos não otimizados para um algoritmo otimizado. Na perspectiva dos inquilinos, a virtualização é uma operação transparente. Algumas tecnologias tornaram possível o desenvolvimento de tal virtualização. O *Explicit Congestion Notification* (ECN) é uma técnica de identificação de problemas de congestionamento que atua observando o limite de ocupação das filas através do mecanismo de gerenciamento de fila *Random Early Detection* (RED). Embora a Virtualização do Controle de Congestionamento (VCC) apresente uma promissora utilização, poucos estudos foram realizados com aplicações reais. Dentre as aplicações existentes, o *Hadoop MapReduce* (HMR) é amplamente utilizado, possuindo, sobretudo, fluxos internos com volume e periodicidade distintos. Nesse contexto, o presente trabalho caracterizou o impacto que a VCC causa no tráfego do HMR executado em cenários onde aplicações com pilha TCP otimizados e não otimizadas estão presentes, competindo pelos recursos de rede. A análise experimental discute o tempo de execução da aplicação, o comportamento das filas nos equipamentos intermediários e o tempo para completar os fluxos individuais. Constatou-se que o HMR tem seu desempenho fortemente influenciado pelo volume de dados de aplicações atualizadas.

Palavras-chaves: Virtualização, Controle de Congestionamento, TCP, data center, *Hadoop*, *MapReduce* e IaaS.

ABSTRACT

The management and financial difficulties that companies face in maintaining a technology park have contributed to the emergence of the IaaS providers that offer virtualized services of processing, storage and communication. This gives rise to DCs of computational clouds that perform tasks for resource management while tenants focus on business activity. Cloud providers host multiple tenant virtual machines (MVs), configured with distinct versions of SOs, optimized and non-optimized TCP libraries and algorithms. The variety of algorithms and configurations related to the TCP protocol constitutes a heterogeneous communication scenario. Optimized congestion control algorithms tend to be more aggressive in the competition for bandwidth sharing and recover from losses caused by congestion. In this way, applications that use a non-optimized algorithm have a hostile scenario that can make communication unfeasible. This situation of lack of equity in the flows between different tenants affects the sense of justice in the use of the network, indicating to customers a misconception about the quality of the service offered by the provider. Because of the full managerial control in DC, providers can apply congestion control virtualization to translate non-optimized algorithms into an optimized algorithm. From the perspective of tenants, virtualization is a transparent operation. Some technologies have made possible to develop such virtualization. ECN is a congestion problem identification technique that acts by observing the occupancy limit of the queues through the queue management mechanism RED. Although VCC exhibits promising use, few studies have been conducted with actual applications. Among the existing applications, HMR is widely used, having above all internal flows of different volume and periodicity. In this context, the present work characterizes the impact that VCC can cause in Hadoop MapReduce traffic executed in scenarios where optimized and non-optimized TCP stack applications are present competing for network resources. The experimental analysis discusses the execution time of the application, the behavior of the queues in the intermediate equipment and the time to complete the individual flows. It was found that the HMR has its acting greatly influenced by the volume of updated data applications.

Key-words: *Virtualization*, *Congestion Control*,TCP, data center, *Hadoop*, *MapReduce* and IaaS.

LISTA DE ILUSTRAÇÕES

Figura 1 – Fluxo de tarefas do MapReduce, adaptado de (NEVES; ROSE; KATRINIS, 2015). 21
Figura 2 – Fluxo de dados de uma aplicação Hadoop MapReduce, adaptado de (NEVES; ROSE; KATRINIS, 2015). 23
Figura 3 – Topologias de rede, adaptado de (NOORMOHAMMADPOUR; RAGHAVENDRA, 2017). 26
Figura 4 – Arquitetura do Open vSwitch Adaptado de (PFAFF et al., 2015). . . 32
Figura 5 – Operações do TCP. 34
Figura 6 – Informações sobre ECN nos cabeçalhos das camadas 3 e 4. 37
Figura 7 – Arquitetura para virtualização do controle de congestionamento, adaptado de (CRONKITE-RATCLIFF et al., 2016). 43
Figura 8 – Trecho de código com o comando de ativação da VCC e do ECN. . 44
Figura 9 – Fluxograma de ativação do VCC guiado pelas configurações fornecidas via *Sysctl*. 45
Figura 10 – Diagrama da Arquitetura do MRemu, adaptado de (NEVES; ROSE; KATRINIS, 2015). 54
Figura 11 – Topologia do Cenário Origrinal. 54
Figura 12 – Interpretação de um diagrama de caixa, adaptado de (BOXPLOT, 2018). 58
Figura 13 – Tempo de execução do HMR compartilhando o gargalo com um par cliente-servidor de tráfego de *background*. 59
Figura 14 – Perdas de pacotes com o HMR compartilhando o gargalo com um par cliente-servidor de tráfego de *background*. 61
Figura 15 – Formação de filas com o HMR compartilhando o gargalo com um par cliente-servidor de tráfego de *background*. 63
Figura 16 – Tempo de execução do HMR compartilhando o gargalo com quatro pares cliente-servidor de tráfego de *background*. 66
Figura 17 – Perda de pacotes com o HMR compartilhando o gargalo com quatro pares cliente-servidor de tráfego de *background*. 68
Figura 18 – Formação de fila com o HMR compartilhando o gargalo com quatro pares cliente-servidor de tráfego de *background*. 70
Figura 19 – Tempo de execução do HMR e perda de pacotes variando o número de pares cliente-servidor de tráfego de *background*. 72
Figura 20 – Formação de filas variando o número de pares executando tráfego de *background*. 75

LISTA DE TABELAS

Tabela 1 – Trabalhos relacionados com virtualização e gerenciamento de redes em DCs. 46
Tabela 2 – Trabalhos relacionados com aplicação HMR. 48
Tabela 3 – Trabalhos relacionados com Técnicas de Isolamento e Gerenciamento de Fluxo. 49
Tabela 4 – Configurações RED para marcação nas filas dos *switches*. 56
Tabela 5 – Configuração dos experimentos realizados 57
Tabela 6 – Principais observações no experimento 1. 64
Tabela 7 – Principais observações no experimento 2 71
Tabela 8 – Principais observações no experimento 3. 77
Tabela 9 – Recomendações de uso do VCC para aplicações HMR. 80

LISTA DE SIGLAS E ABREVIATURAS

ACDC Administração Centralizada de Controle de Congestionamento

ACK *Acknowledgments*

CDF *Cumulative Distributed Function*

CWND *Congestion Window*

CWR *Congestion Window Reduced*

DC *data center*

DCTCP *Data Center TCP*

ECE *Congestion Experienced*

ECMP *Equal-Cost Multi-Path Routing*

ECN *Explicit Congestion Notification*

ECT *ECN-Capable Transport*

FIFO *First In First Out*

HDFS *Hadoop Distributed File System*

HMR *Hadoop MapReduce*

IaaS Infraestruturas como Serviço

LaaS Links como Serviço

MSS *Maximum Segment Size*

MV Máquina Virtual

NFV *Network Function Virtualization*

OSI *Open System Interconnection*

POD Agrupamento

RED *Random Early Detection*

RTT *Round Trip Time*

RPC *Remote Procedure Call*

RWND *Receiver Window*

SACK *Selective Acknowledgments*

SDN *Software Defined Network*

SLA Acordo de Nível de Serviço

SO sistema operacional

SQL *Structured Query Language*

SSH *Secure Shell*

TCP *Transmission Control Protocol*

TOR *Top of Rack*

VCC Virtualização do Controle de Congestionamento

VI *Virtual Infrastructure*

VLAN *Virtual Local Network*

VPN *Virtual Private Network*

VPS *Virtual Private Server*

SUMÁRIO

1 INTRODUÇÃO 15
1.1 METODOLOGIA 17
1.2 OBJETIVO GERAL 18
1.2.1 Objetivos Específicos 18
1.2.2 Principais contribuições do trabalho 18
1.3 ORGANIZAÇÃO DO TRABALHO 19

2 FUNDAMENTAÇÃO TEÓRICA 20
2.1 MAPREDUCE 20
2.2 HADOOP 21
2.3 DCs PARA EXECUÇÃO DE HMR 24
2.3.1 Tráfego HMR em DCs 24
2.3.2 Topologias de DC 25
2.3.2.1 Topologia Fat-tree 27
2.3.2.2 Topologia VL2 27
2.3.2.3 Topologia Jellyfish 28
2.3.2.4 Topologia Leaf-Spine 28
2.3.2.5 Topologia BCube 28
2.3.2.6 Topologia Dcell 29
2.3.2.7 Considerações sobre as Topologias 29
2.3.3 Virtualização de DCs 30
2.3.3.1 Máquina Virtuals (MVs) e Contêineres 30
2.3.3.2 Redes Virtuais 31
2.4 CONTROLE DE CONGESTIONAMENTO 33
2.4.1 Controle de Congestionamento nas Extremidades da Rede . . . 33
2.4.2 Controle de Congestionamento com Auxílio do Núcleo 35
2.4.3 Limitações do controle de congestionamento em DCs 38
2.5 CONSIDERAÇÕES PARCIAIS 39

3 VIRTUALIZAÇÃO DO CONTROLE DE CONGESTIONAMENTO (VCC) 41
3.1 CONCEITOS E DEFINIÇÕES 41
3.2 IMPLEMENTAÇÃO DO CONTROLE DE CONGESTIONAMENTO . . 42
3.3 TECNOLOGIAS E TRABALHOS RELACIONADOS 45
3.3.1 Virtualização e Gerenciamento de Redes em DCs 46
3.3.2 Otimização e Gerenciamento de Aplicações HMR 48

3.3.3 Outras Técnicas de Isolamento e Gerenciamento de Fluxo 49
3.3.4 Discussão 50
3.4 CONSIDERAÇÕES PARCIAIS 51

4 CARACTERIZAÇÃO DO DESEMPENHO DO HMR SOBRE VCC . . 52
4.1 MÉTRICAS PARA CARACTERIZAÇÃO 52
4.2 EMULAÇÃO DO TRÁFEGO HMR 52
4.2.1 Emulador MRemu 53
4.2.2 Rastro de Execução e Ambiente Hospedeiro 54
4.3 CENÁRIOS EXPERIMENTAIS 55
4.3.1 Variantes do TCP 55
4.3.2 Configuração da Marcação de Pacotes 55
4.3.3 Tráfego de *Background* 56
4.4 RESULTADOS EXPERIMENTAIS E CARACTERIZAÇÃO 57
4.4.1 Análise do Experimento 1 58
4.4.1.1 Tempo de Execução 58
4.4.1.2 Perda de Pacotes 60
4.4.1.3 Formação de Filas 62
4.4.1.4 Principais Observações 64
4.4.2 Análise do Experimento 2 64
4.4.2.1 Tempo de Execução 65
4.4.2.2 Perda de Pacotes 67
4.4.2.3 Formação de Filas 69
4.4.2.4 Principais Observações 71
4.4.3 Análise do Experimento 3 71
4.4.3.1 Tempo de Execução 72
4.4.3.2 Perda de Pacotes 73
4.4.3.3 Formação de Filas 74
4.4.3.4 Principais Observações 76
4.5 DISCUSSÃO E PRINCIPAIS OBSERVAÇÕES 76
4.6 CONSIDERAÇÕES FINAIS 80

5 CONSIDERAÇÕES E PERSPECTIVAS 82
5.1 SUGESTÕES DE TRABALHOS FUTUROS 82
5.2 PUBLICAÇÕES REALIZADAS 83

REFERÊNCIAS 84

1 INTRODUÇÃO

A rapidez com a qual a evolução tecnológica acontece nos dias atuais tem tornado difícil para as empresas manterem atualizados os recursos de hardware e software necessários para o andamento de seus negócios. Contratar um terceiro para realizar esse trabalho se tornou algo comum (ROSENBERG; MATEOS, 2010). Dessa forma as empresas se concentram nas atividades que dizem respeito ao negócio e o aparato tecnológico é terceirizado. A necessidade de conectividade com serviços de qualquer lugar e a qualquer momento levou ao crescimento das nuvens computacionais. Serviços como processamento, armazenamento e infraestrutura, são oferecidos totalmente personalizados sem interferência nas configurações realizadas pelos clientes. Essa maleabilidade gerencial oferecida pelo provedor de Infraestruturas como Serviço (IaaS) é uma das principais características das nuvens computacionais (MELL; GRANCE, 2011).

Os *data center*s (DCs) de nuvens computacionais IaaS se tornaram um ambiente no qual múltiplos inquilinos hospedam aplicações que divergem em necessidades tecnológicas e configurações. Neste cenário de múltiplas tecnologias de comunicação, fluxos sensíveis à latência, assim como fluxos que exigem alta vazão passam a coexistir no mesmo ambiente. Especificamente, uma Máquina Virtual (MV) pode ser composta com diferentes configurações de memória, armazenamento e processamento. Sobretudo, o cliente possui total liberdade para controlar o sistema operacional (SO), atuando na instalação e atualização das aplicações, bibliotecas e núcleo gerencial.

A complexidade na manutenção e as dependências de versões específicas (bibliotecas e SOs) constituem fatores limitantes para a atualização das MVs hospedeiras. Em SOs não otimizados, os algoritmos do *Transmission Control Protocol* (TCP) para controlar e evitar congestionamento estão aquém dos últimos avanços. Algoritmos para controle auxiliam na recuperação do desempenho na ocorrência de gargalos, enquanto algoritmos para evitar congestionamento atuam na predição de eventuais perdas (CHIU; JAIN, 1989). Tradicionalmente, o TCP controla e evita congestionamentos nas extremidades da rede (JACOBSON, 1988), sem suporte nativo dos equipamentos que compõem o núcleo, baseado apenas em informações inferidas sobre os fluxos de dados.

O algoritmo TCP sofreu várias alterações ao longo do tempo para acompanhar a evolução da comunicação em ambientes computacionais. Algoritmos como *Slow Start*, *Fast Retransmit* e *Fast Recovery* foram implementados e melhoraram a forma de identificação e recuperação de uma situação de perda causada por congestiona-

mento (STEVENS, 1997), (FLOYD; HENDERSON; GURTOV, 2004). Conexões TCP iniciadas com diferentes mecanismos de controle de congestionamento contribuem para que ocorra o desbalanceamento da equidade no compartilhamento dos recursos utilizados pelos inquilinos. Aqueles com mecanismos otimizados absorvem maior parcela de largura de banda, podendo comprometer o Acordo de Nível de Serviço (SLA) dos inquilinos que utilizam mecanismos não otimizados (POPA et al., 2012).

Diversas versões do TCP foram propostas para otimizar o desempenho do tráfego em DCs (WU et al., 2012; MITTAL et al., 2015; KUZMANOVIC, 2005; ALIZADEH et al., 2010) alterando a interpretação do *feedback* binário originalmente concebido (CHIU; JAIN, 1989). Em alguns casos, o núcleo da rede participa do controle de congestionamento oferecendo marcações de pacotes com alertas de possíveis gargalos (KUZMANOVIC, 2005). Entretanto, os esforços realizados esbarram na execução de SOs não otimizados (JUDD, 2015; HOFMANN; WOODS, 2010). Ou seja, embora o DC execute um algoritmo específico para controle de congestionamento, algoritmos não otimizados (*i.e.*, sem suporte à interpretação e marcação de pacotes) ou com configurações agressivas (*i.e.*, tamanho de *buffers*, comportamente do *slow start*), comprometem o desempenho das aplicações finais (CRONKITE-RATCLIFF et al., 2016; HE et al., 2016). Portanto, o problema da desigualdade na obtenção da parcela de banda ainda permanece enquanto inquilinos com TCP distintos compartilharem o uso do DC.

De fato, é interesse do provedor de IaaS contornar o problema imposto pelo cenário heterogêneo de algoritmos de controle de congestionamento, qualificando o serviço ofertado aos clientes. Possíveis soluções compreendem a reserva de recursos de comunicação (largura de banda e configuração dos *switches*) (SOUZA et al., 2017; ZAHAVI et al., 2016) e configuração dinâmica de filas de encaminhamento (JUDD, 2015). Entretanto, as técnicas elencadas tendem a subutilizar os recursos de comunicação ou possuem elevada complexidade de gerenciamento (POPA et al., 2012).

Nesse contexto, a VCC foi proposta como uma alternativa para uniformizar os algoritmos de controle em execução nas MVs (CRONKITE-RATCLIFF et al., 2016; HE et al., 2016). Em resumo, os provedores possuem acesso administrativo aos *switches* virtuais ou hipervisores de MVs, responsáveis por encaminhar os pacotes entre as MVs dos inquilinos. Assim, uma camada de virtualização pode interceptar o tráfego não padronizado (otimizado ou agressivo) e manipular, quando necessário, para atender aos requisitos dos protocolos atualizados do DC. É importante salientar que nenhuma alteração é realizada na pilha TCP nas MVs dos inquilinos. A virtualização procura uniformizar os fluxos gerados pelos múltiplos inquilinos utilizando o serviço do provedor de IaaS.

As técnicas de virtualização utilizadas neste trabalho são baseadas em pro-

postas (CRONKITE-RATCLIFF et al., 2016; HE et al., 2016) estudadas com cargas controladas, que representam somente um cenário em DCs. Originalmente, a Virtualização do Controle de Congestionamento (VCC) foi analisada com cargas sintéticas indicando a eficácia para cenários que objetivavam a equidade do compartilhamento e maximização da utilização em gargalos para fluxos majoritariamente longos (CRONKITE-RATCLIFF et al., 2016; HE et al., 2016). Os resultados obtidos demonstraram que em configurações adequadas a VCC pode melhorar a equidade em comunicações com controle de congestionamento heterogêneos. O processamento da tradução envolve as atividades de abrir o pacote, analisar o cabeçalho e introduzir informações de *Explicit Congestion Notification* (ECN), entretanto, os autores apontaram que a sobrecarga computacional gerada no processo de tradução é inferior a 1%. Porém, cargas de comunicação reais de aplicações tradicionalmente executada em DCs de nuvens não foram analisadas.

Dentre as aplicações hospedadas em DCs de nuvens, o HMR é uma aplicação utilizada para manipular grande volumes de dados como ferramentas e serviços para conteúdo digital, comércio eletrônico, serviço de rede social, empresa jornalística mundial, provedores de busca, etc (HADOOP, 2018). Assim, por ser amplamente utilizado em nuvens (ALIZADEH et al., 2010) (THUSOO et al., 2010), a aplicação *Hadoop MapReduce* (HMR) foi selecionada como alvo do presente estudo. Especificamente, aplicações baseadas em HMR possuem fluxos de comunicação com características distintas das inicialmente estudadas pelos desenvolvedores de VCC (ALIZADEH et al., 2010; ROY et al., 2015).

Em suma, o objetivo do presente trabalho é analisar a aplicabilidade de VCC para aplicações HMR. Para entender o impacto que a virtualização do controle de congestionamento causa em aplicações reais, foi utilizado a ferramenta *MRemu* (NEVES; ROSE; KATRINIS, 2015) para geração de tráfego de dados competindo com fluxos de diferentes tamanhos.

1.1 METODOLOGIA

O estudo é motivado pela diversidade de fluxos de comunicação (número e volume de dados) presentes em ambiente de rede do DC (ALIZADEH et al., 2010; CHOWDHURY et al., 2011). Originalmente, a VCC foi analisada com cargas sintéticas controladas, indicando a eficácia para cenários que objetivavam a equidade do compartilhamento e maximização da utilização em gargalos.

Conforme (KOCHE, 1997), o objetivo da pesquisa exploratória, realizada no presente trabalho, é ampliar o conhecimento do pesquisador a respeito de um tema. Para fundamentação teórica, a execução de pesquisa exploratória foi realizada atra-

vés de pesquisa bibliográfica identificando as principais contribuições teóricas sobre o assunto publicadas em livros, artigos, enciclopédias, etc. A pesquisa nos mecanismos de busca acadêmica foi realizada considerando artigos que tivessem no *título* e no *abstract* alguma das palavras chave: virtualização, controle de congestionamento, TCP, data center, *Hadoop*, *MapReduce* e *IaaS*. Posteriormente, uma análise técnica e experimental foi realizada sobre VCC e a aplicação alvo.

1.2 OBJETIVO GERAL

O objetivo do presente trabalho é caracterizar o impacto da virtualização do controle de congestionamento na execução do HMR em DCs com múltiplos clientes, contendo um conjunto heterogêneo de aplicações executadas simultaneamente.

1.2.1 Objetivos Específicos

- Pesquisar os algoritmos de controle de congestionamento e como funcionam;
- Estudar o controle de congestionamento nas extremidades da rede;
- Estudar as tecnologias de controle de congestionamento no núcleo da rede (ECN e RED);
- Compreender o funcionamento da ferramenta Mininet;
- Estudar o HMR e MRemu;
- Realizar testes representando gargalos, utilizando a ferramenta Mininet para simular DCs reais; e
- Coletar, documentar e analisar os dados.

1.2.2 Principais contribuições do trabalho

Como contribuição, o presente trabalho: *(i)* analisa o tempo de execução da aplicação HMR utilizando rastros reais de execuções, considerando a presença no DC compartilhado, de pilhas TCP otimizadas, legadas e virtualizadas; *(ii)* discute o impacto das configurações de marcação de pacotes para prevenção de congestionamento sobre HMR; e *(iii)* correlaciona a formação de filas e o descarte de pacotes com o tempo de execução de HMR. Sobretudo, a caracterização indica que aplicações baseadas em HMR sofrem um impacto profundo quando executadas em ambientes com VCC. É importante ressaltar que a caracterização é direcionada para a comunicação, e não para o processamento da operações de mapeamento e redução da aplicação HMR.

1.3 ORGANIZAÇÃO DO TRABALHO

O trabalho está estruturado da seguinte maneira: No Capítulo 2 é apresentada uma revisão de literatura abordando as diferentes topologias de rede e o controle de congestionamento do TCP, sobretudo em ambiente de DCs. Ainda, aprofunda o entendimento de como ocorre o controle de congestionamento nas extremidades e no núcleo da rede. A aplicação alvo, HMR, é revisada no mesmo capítulo. Por fim, neste capítulo são discutas as limitações do controle de congestionamento em DC. Posteriormente, no Capítulo 3, são abordadas a definição e conceito de virtualização de controle de congestionamento, bem como as tecnologias e trabalhos relacionados. Na sequência é discutido a implementação da VCC. No Capítulo 4 é realizada uma análise experimental do desempenho de HMR em DCs analisando o desempenho de acordo com três ambientes: TCP não otimizado, TCP otimizado e VCC. Por fim, no Capítulo 5, são apresentadas as considerações finais e iniciativas para continuidade e avanço na pesquisa sobre o assunto.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são abordados conceitos necessários para a compreensão dos elementos que fazem parte deste trabalho. Inicialmente, na Seção 2.1, é explicado o *MapReduce*, um paradigma de programação que faz uso intensivo da programação paralela e sistemas distribuídos. Na sequência, na Seção 2.2, é detalhado o funcionamento do *Hadoop MapReduce* (HMR). Em seguida, na Seção 2.3 são descritas as topologias de rede em ambientes de DCs, incluindo a revisão sobre virtualização de recursos computacionais e de comunicação. O controle de congestionamento do TCP é revisado na Seção 2.4. Na Seção 2.4.3 são apresentados os desafios e limitações relacionados com o controle de congestionamento em DCs. Finalizando este capítulo, na Seção 2.5 são apresentadas as considerações parciais.

2.1 MAPREDUCE

O volume de dados gerados diariamente por aplicações corporativas, serviços e sistemas web, como mídias sociais, comércio eletrônico, entre outras ultrapassou a casa de Petabytes. Por exemplo, em 2007 o *Facebook* produziu cerca de 15 Terabytes de dados, 3 anos depois, em 2010 esse volume aumentou para 700 Terabytes (THUSOO et al., 2010). Esses dados são armazenados de forma não estruturada e processados em sistemas e linguagens diferentes, em alguns casos sujeitos a problemas de compatibilidade.

Uma das dificuldades inerentes a esses dados é dar um significado para eles, extraindo uma informação de valor (GOLDMAN et al., 2018). Neste contexto, surgiu o termo *Big Data* que se refere não somente ao volume de dados gerados, mas também a diversidade de aplicações e a velocidade com que esses dados são processados (IBM, 2018). Sobretudo, o *Facebook* indicou que $1/3$ do tempo de processamento é devido às transmissões de dados. Devido a escalabilidade oferecida por serviços IaaS essas aplicações tem sido executadas em ambientes de nuvem.

Dentre as possíveis alternativas para processamento de *BigData*, o MapReduce é um conceito de programação que busca dividir o processamento de uma aplicação entre um aglomerado de servidores. É constituído de duas funções: *Map* e *Reduce*, presentes na programação funcional. Uma função *Map* recebe um conjunto de dados para processar, normalmente armazenados em um sistema de arquivos distribuídos. O resultado dessa execução é a entrada para a fase de redução. Esse processamento posteriormente é agrupado e o resultado disponibilizado ao cliente.

MapReduce foi inicialmente implementado em *LISP*, sendo que, posterior-

mente a *Google* aplicou amplamente este conceito e o paradigma em Java, C++ e Python (GOLDMAN et al., 2018).

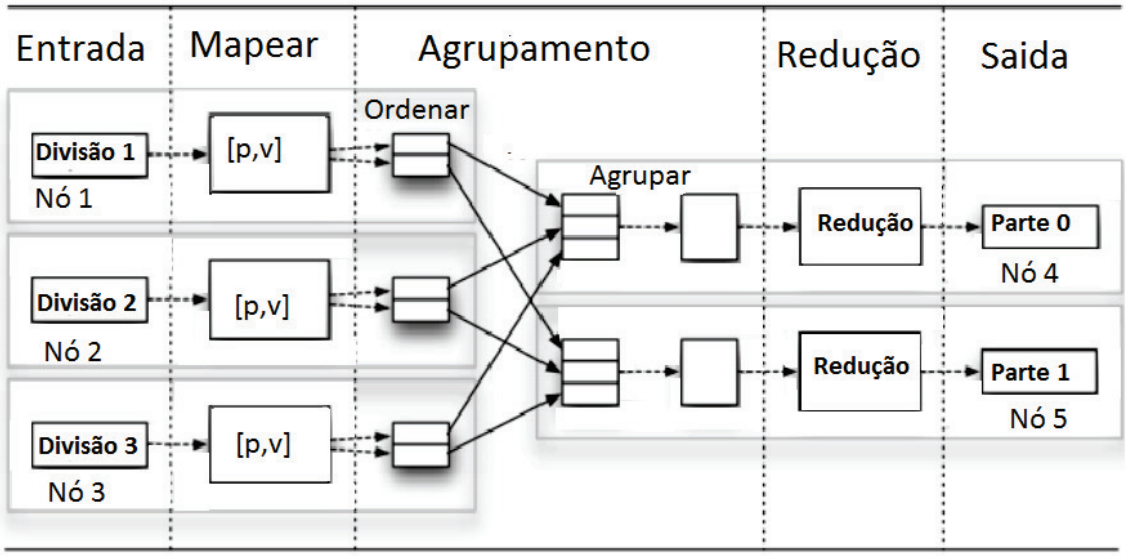


Figura 1 – Fluxo de tarefas do MapReduce, adaptado de (NEVES; ROSE; KATRINIS, 2015).

Inicialmente, conforme representado na Figura 1, os dados de entrada são divididos entre os nós do aglomerado de servidores (ou MVs), e posteriormente mapeados em conjuntos de pares valorados *[p, v]*. A etapa de agrupamento realiza uma ordenação inicial e uma posterior organização dos dados. Essa fase, segundo (NEVES; ROSE; KATRINIS, 2015), inicia imediatamente quando aproximadamente 5% da fase de mapeamento está concluída. Por fim, é realizada a redução, agrupando os resultados e preparando a saída (ou próxima rodada de processamento).

2.2 HADOOP

É um conjunto de ferramentas para suportar o processamento de *MapReduce*, construídas e distribuídas de acordo com o modelo de licenciamento *Apache*. Alguns exemplos de ferramentas que fazem parte deste conjunto são listados na sequência:

- **Avro** (AVRO, 2018), serializa dados em padrão binário baseados em *schemas*. Suporta chamadas *Remote Procedure Call* (RPC).
- **Chukwa** (CHUKWA, 2018), ferramenta especializada em análise de *logs* do sistema. Utiliza *Hadoop Distributed File System* (HDFS) para armazenar os dados e o *MapReduce* para análise dos dados.
- **Hbase** (HBASE, 2018), banco de dados distribuído e escalável para armazenar dados estruturados em grandes tabelas.

- **Hive** (HIVE, 2018), desenvolvido pela Facebook, é uma linguagem de consulta similar ao *Structured Query Language* (SQL).
- **Pig** (PIG, 2018), um compilador capaz de transformar programas em sequências *Mapreduce*.
- **ZooKeeper** (ZOOKEEPER, 2018), criado pelo Yahoo. Coordena atividades em aplicações distribuídas de alto desempenho como configuração de nós, sincronização de processos e distribuição de serviços.
- **HMR** (GOLDMAN et al., 2018), Divide o processamento de uma tarefa entre os nós de um aglomerado de servidores, mapeando em um conjunto de [par,valor]. Posteriormente, submete o trabalho a um processo de redução, gerando o resultado final.

Dentre os componentes, o presente trabalho focou especificamente no HMR. Algumas vantagens do HMR são: *(i)* código aberto, mantido por uma comunidade ativa; *(ii)* economia, ou seja não é preciso licenças de *software*, e o aglomerado de servidores não necessita de *hardware* especializado; *(iii)* robustez: oferece replicação de dados, metadados e informações de processamento para mitigar os efeitos da falhas no *hardware*; *(iv)* a escalabilidade é limitada pelos recursos do aglomerado de servidores. A expansão não demanda alterações no código da aplicação. *(v)* simplicidade, o *software* abstrai questões relacionadas à computação paralela, como tolerância a falhas, balanceamento de carga e escalonamento (GOLDMAN et al., 2018).

Internamente, HMR é constituído de duas partes: *(i)* o *Hadoop Distributed File System* (HDFS) que é responsável pelo armazenamento dos dados e, *(ii) MapReduce* que é um modelo de programação responsável pela análise dos dados (DEAN; GHEMAWAT, 2008). Ainda, a ferramenta HMR é constituída de 5 componentes: *NameNode* cuja função é mapear a localização dos dados, dividir arquivos em blocos, obter metadados dos arquivos e gerenciar réplicas. *SecondaryNameNode* é um auxiliar do *NameNode* e faz checagem periódica do serviço oferecendo uma alternativa caso ocorra falha. *DataNode* armazena os blocos de dados. Cada instância pode armazenar vários blocos inclusive de aplicações diferentes. Estes 3 componentes pertencem ao HDFS.

A Figura 2 representa os outros componentes, que são o *JobTracker* e o *TaskTracker*. O primeiro é responsável pelo gerenciamento das tarefas a serem executadas pelo HMR, identificando em qual nó uma tarefa será executada e monitorando a execução, transferindo para outro nó na ocorrência de falha. O segundo componente é o *TaskTracker* que é o componente que executa a tarefa. Os componentes *NameNode*, *SecondaryNameNode* e *JobTracker* são únicos para toda a aplicação. O *Task-*

Tracker e o *DataNode* são instanciados em cada máquina participante do aglomerado onde a tarefa será executada.

O HMR otimiza a comunicação de dados agendando o processamento em servidores próximo dos dados a serem processados. Para isso utiliza o HDFS, um sistema de arquivos adequado para ambientes de baixo custo que consegue garantir confiabilidade criando várias réplicas do arquivo de dados (GUNARATHNE et al., 2010). O servidor principal, chamado *Master*, tem uma visão dos metadados de toda a árvore de arquivos do sistema e gerencia a distribuição das tarefas para o servidores que processam os dados, os *Workers*. O *Master* monitora a execução nos *Workers* e define as tarefas que cada um executará, seja a tarefa de *mapeamento* ou *redução* (KATAL; WAZID; GOUDAR, 2013).

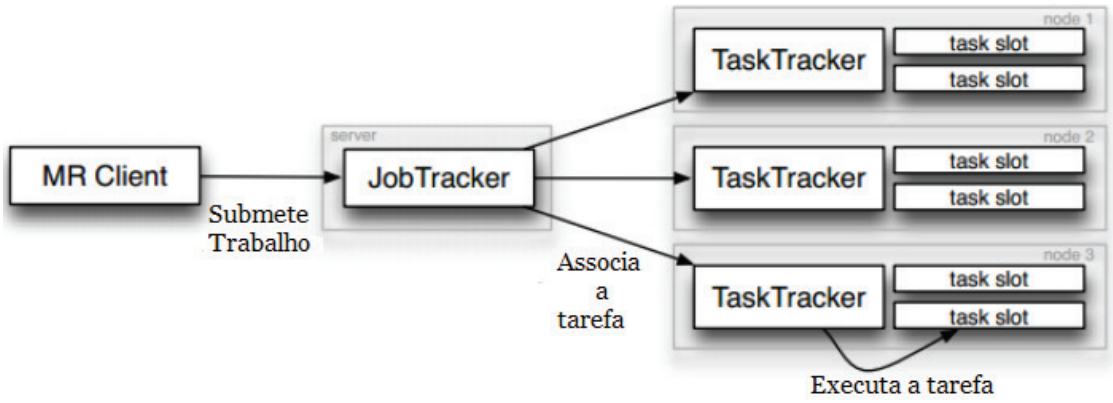


Figura 2 – Fluxo de dados de uma aplicação Hadoop MapReduce, adaptado de (NEVES; ROSE; KATRINIS, 2015).

Para auxiliar no entendimento sobre o tráfego de dados da aplicação, a Figura 2 ilustra o fluxo interno do HMR. *(i)* Inicialmente um cliente solicita um serviço. O *Master* ativa o *NameNode* identificando a localização dos dados e a tarefa a ser executada. *(ii)* O *JobTracker* recebe esta solicitação e identifica os *Workers* que podem processar as requisições. *(iii)* As tarefas são distribuídas entre os servidores participantes do aglomerado, nos quais serão executadas. São instanciados o *DataNode* e *TaskTracker* em cada *Worker*.

É importante ressaltar que 33% do tempo de execução do HMR é atribuído à tarefas de comunicação (CHOWDHURY et al., 2011), motivando a realização do presente trabalho, onde foi proposto caracterizar o impacto da comunicação em ambientes heterogêneos. Métricas relacionadas ao processamento dos dados do HMR são irrelevantes para o escopo deste trabalho.

Por fim, como DCs de nuvens IaaS hospedam aplicações com padrões distintos de comunicação, àquelas baseadas em particionamento, processamento e agregação de informações trafegam dados de controle sensíveis à latência, bem como

tráfego de fluxos para sincronização de massas de dados. Ou seja, os fluxos sensíveis à latência concorrem com o tráfego de *background*, constituído por pequenos e grandes fluxos. Analisando a ocupação dos *buffers* em *switches* e servidores, fluxos maiores que 1MB possuem baixa multiplexação e consomem grande parcela do espaço disponível (ALIZADEH et al., 2010). Consequentemente, induzem a formação de filas e o aumento da latência para os demais fluxos.

2.3 DCs PARA EXECUÇÃO DE HMR

Um DC é formado por sala em que são colocados elementos computacionais como, por exemplo, servidores, equipamentos de armazenamento e equipamentos de rede que podem oferecer serviços de processamento, armazenamento e comunicação de dados (BANI-AHMAD; SA'ADEH, 2017). Por sua vez, nuvens IaaS são estruturas que permitem ao inquilino contratar recursos computacionais, tais como processamento, armazenamento e comunicação em rede, mantendo o controle sobre sistemas operacionais e outros elementos relacionados às suas aplicações, porém sem preocupações no que se refere aos aspectos de gerenciamento da rede subjacente (MELL; GRANCE, 2011).

A possibilidade de hospedar aplicações em um ambiente com infraestrutura necessária para sua execução, sem ter preocupações com questões de implantação, manutenção e gerenciamento destes recursos operacionais, tem sido uma das motivações para inquilinos hospedarem suas aplicações em nuvem. Por outro lado, o provedor de serviços precisa oferecer uma configuração da infraestrutura: redes, roteamento, processamento, armazenamento, servidor de atribuição de endereços, resolução de nomes, etc., necessários para atender aos diferentes requisitos de aplicações heterogêneas hospedadas por vários inquilinos. Como aplicações tem necessidades diferentes de comunicação, algumas requerem alta vazão, enquanto outras necessitam de baixa latência, a rede pode ser configurada de acordo com a topologia mais adequada para atender a essas necessidades.

2.3.1 Tráfego HMR em DCs

Provedores de nuvem tem investido na infraestrutura de seus *data center*s (DCs) no intuito de garantir a operacionalização dos serviços que oferecem. A concentração de estruturas virtualizadas, em um pequeno espaço físico, atendendo uma diversidade de aplicações tem desafiado a forma como os recursos podem ser interligados. Essa realidade tem impactado o projeto de redes, inclusive no que se refere a topologia.

Por exemplo, (ROY et al., 2015) analisaram o *DC* do *Facebook* e avaliaram

o comportamento de 3 padrões distintos de tráfego: *(i)* aplicação *Hadoop MapReduce* (HMR); *(ii)* máquinas executando serviços de requisições web e *(iii) cache* de dados. Os autores constataram que a escolha da topologia depende da demanda de dados dos serviços que oferecem. Ainda, identificaram uma tendência de baixa utilização da rede nas bordas, enquanto que nas camadas de agregação e núcleo ocorre uma grande demanda, criando pontos de congestionamento. Alguns padrões tem um tráfego mais intenso intra-rack enquanto que outros dependem de comunicação com outros *clusters* em diferentes *racks*. O tráfego de dados do HMR neste cenário, ocorre predominantemente em rajada curtas, apresentando uma variação de demanda através dos servidores, bem como em relação ao tempo.

Alguns desafios a que aplicações de mapeamento e redução implementados pelo HMR sendo executadas em nuvem são elencadas por (GUNARATHNE et al., 2010): *(i)* essas aplicações são fortemente influenciadas pelo local onde os dados e metadados estão armazenados. A estrutura de armazenamento deve ser escalável e bem estruturada a fim de evitar pontos de falhas e gargalos na comunicação; *(ii)* Em ambientes virtualizados estão sujeitas a carga da rede subjacente e operações de I/O tornando a performance da rede inconsistente; Pelo fato de compartilhar a rede com outras aplicações, sofrem de oscilações no desempenho e dificuldade para escalar; *(iii)* Como a aplicação é distribuída entre *clusters* sujeitos à falha, a recuperação quando um *host* apresenta problemas é crucial para a continuidade da tarefa; *(iv)* A escolha da instância de MV que oferece a melhor relação custo beneficio, ainda ter acesso armazenamento definitivo de *logs* das operações após o encerramento da instância da MV.

A análise da comunicação em DC realizada por (ALIZADEH et al., 2010), mostra um padrão do tipo *Particionamento/Agregação* que pode ocorrer em vários níveis, hierarquicamente relacionados, no qual a aplicação divide a execução da tarefa entre várias executores, os *agregadores* e os *workers*. O processamento obtido pelo *workers* é repassado aos *agregadores* compilando os resultados parciais obtidos produzindo um resultado final. Por exemplo, uma pesquisa *web* pode ser dividida entre vários *agregadores* e *workers*, cada um responsável por parte da pesquisa. É importante que o tarefa de cada parte envolvida aconteça dentro do prazo estabelecido. Ou seja, atraso no tempo de execução pode comprometer a entrega do resultado final podendo comprometer acordo SLA estabelecidos.

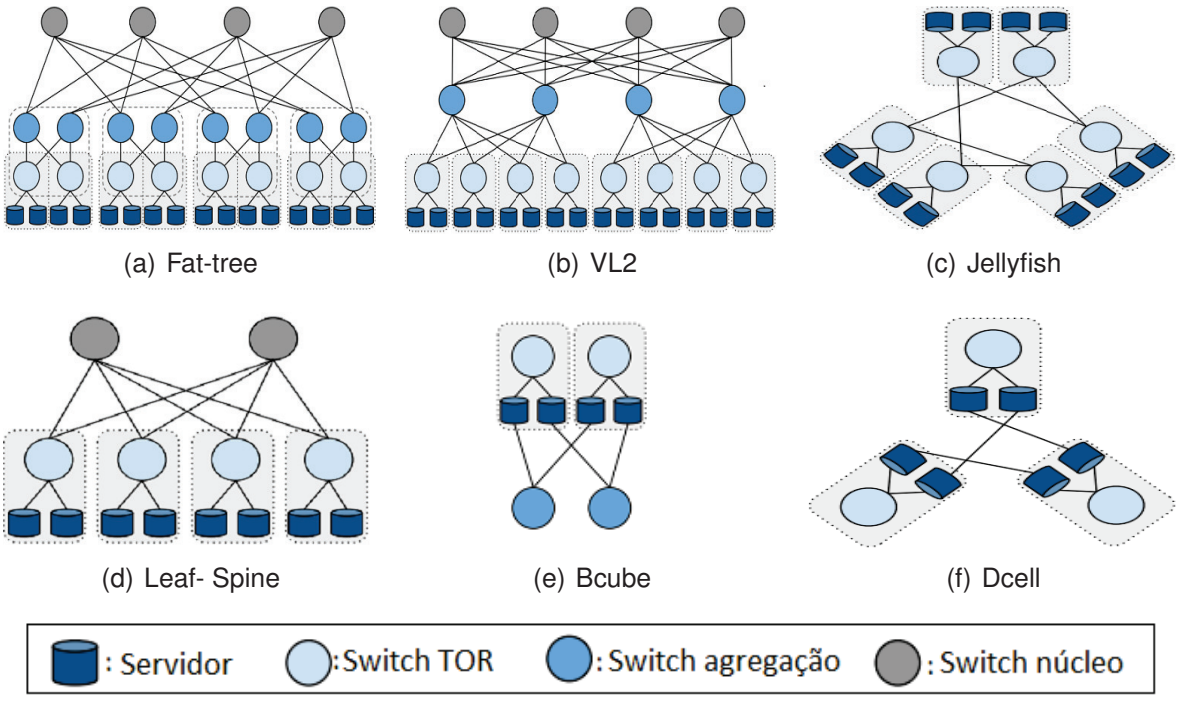
2.3.2 Topologias de DC

A topologia da rede do *DC* influencia em aspectos que dizem respeito à escalabilidade, roteamento, balanceamento de carga, tolerância à falhas e redundância de enlaces e podem ser classificadas como:*(i)* centralizadas no servidor e *(ii)* centraliza-

das nos *switches*.

As topologias centralizadas em *switches* são constituídas de duas ou três camadas e dependem de *switches* corporativos com diferentes números de portas. Estas topologias melhoram o tráfego através dos múltiplos caminhos que formam, possibilitando balanceamento de carga e rotas alternativas. Essa multiplicidade de caminhos também torna a rede mais tolerante a falhas, uma vez que um enlace estiver indisponível, a rede consegue funcionar utilizando outra rota. Por outro lado, essas topologias apresentam escalabilidade limitada e aumento de complexidade de gerenciamento e configuração à medida que aumenta em tamanho.

As topologias centradas no servidor são topologias organizadas em células, com um *switch* interligando servidores dentro da célula. Um *switch* não se conectada diretamente a outro. A comunicação entre células é realizada pelo servidor que é equipado com mais de uma placa de rede (YEDDER et al., 2017). Ou seja, o roteamento é processado no servidor. Nesta seção são apresentadas algumas topologias apontadas pela literatura contemporânea e suas características (resumidas na Figura 3), representando as duas classificações elencadas.



(a) Fat-tree (b) VL2 (c) Jellyfish

(d) Leaf- Spine (e) Bcube (f) Dcell

(g) Legenda

Figura 3 – Topologias de rede, adaptado de (NOORMOHAMMADPOUR; RAGHAVENDRA, 2017).

2.3.2.1 Topologia Fat-tree

Na topologia *Fat-tree*, Figura 3(a), os equipamentos intermediários são separados em três níveis: núcleo, agregação e borda. Cada equipamento no nível de núcleo está interligado aos equipamentos no nível de agregação provendo assim caminhos redundantes entre os dois níveis. Por sua vez, os equipamentos de borda apresentam caminhos redundantes com dispositivos intermediários dentro do mesmo agrupamento e são responsáveis por permitirem que os servidores sejam conectados à rede.

Algoritmos de roteamento devem ser configurados nos *switches* para evitar a formação de laços e balancear a carga entre os caminhos redundantes. Esta topologia é construída com *switches* de baixo custo para conectar servidores, garantindo escalabilidade e alta vazão em qualquer bisecção de rede (GUO et al., 2009). É ainda baseada na topologia *Clos*, um tipo de topologia proposta inicialmente por Charles Clos para interligar um grande número de linhas telefônicas utilizando *switches* distribuídos em vários níveis (AL-FARES; LOUKISSAS; VAHDAT, 2008).

Exemplos de topologias *Fat-tree* são observadas nos DCs do Google. O tráfego dentro do DC é transportado por uma rede interna chamada *B2* com um plano de controle descentralizado. Outro nível, chamado *B4* estabelece a comunicação entre os DCs, focando em uma sistemática centralizada do plano de controle da rede. A diversidade de projeto destes DCs é uma variante da topologia *Clos*. A camada de topo da malha da rede denominada *Spine*, possui *switches* de agregação e núcleo (GOVINDAN et al., 2016). A camada mais baixa é constituída de *switches Top of Rack* (TOR) que provem conectividade para os servidores.

2.3.2.2 Topologia VL2

A topologia *VL2*, representada na Figura 3(b), é uma topologia centralizada no *switch*, formada por 3 níveis, sendo que os enlaces que interligam *switches* têm uma largura de banda maior que os enlaces entre servidores e *switches*. Esta topologia é semelhante a topologia adotada anteriormente por *mainframes* (CISCO, 2007) com a ressalva de utilizar *switches* de baixo custo em uma topologia *Clos* semelhante à topologia *Fat-tree*. Cada *switch* TOR está interligado a 2 *switches* da camada de agregação. Dessa forma, cada *switch* da camada de agregação atende ao fluxo da mesma quantidade de *switches* da camada de acesso. Enquanto cada *rack* suporta entre 20 e 40 servidores. A implementação de roteamento é realizada nos *switches* da camada de núcleo (GREENBERG et al., 2009). Cada *switch* da camada de acesso está conectado a dois *switches* na camada de agregação. Estes por sua vez, estão interligados em conexões redundantes com todos os *switches* da camada de núcleo. Dessa forma, esta rede apresenta tolerância à falhas, com possibilidade de balanceamento de carga

e múltiplas rotas para envio e recebimento de pacotes.

2.3.2.3 Topologia Jellyfish

Na Figura 3(c) é apresentada a topologia *Jellyfish*, uma topologia centralizada no *switch* que é constituída por *switches* TOR, com conexão entre si realizada de forma randômica, verificando a disponibilidade de portas livres. A tolerância à falhas é obtida estabelecendo conexões redundantes entre os *switches*. Segundo (SINGLA et al., 2012) com a mesma bisecção de largura de banda, esta topologia suporta mais servidores e consegue manter a média de vazão mesmo com um número maior de enlaces em estado de falha que a topologia *Fat-tree*. Entretanto, o problema desta topologia reside no fato de que os protocolos de roteamento não são eficazes na escolha do caminho devido as constantes trocas para o próximo salto.

2.3.2.4 Topologia Leaf-Spine

Por sua vez , a topologia *Leaf-Spine* representada na Figura 3(d) é uma topologia de duas camadas, centralizada no *switch*. Essa topologia oferece alta vazão, formando um único bloco no qual todos os *switches* e servidores estão diretamente conectados (ALIZADEH; EDSALL, 2013). Os *switches Leaf* são equipamentos *Top of Rack* (TOR) conectados aos servidores. Cada *switch Spine* da topologia é conectado diretamente a todos os *switches Leaf*, formando um grafo bipartido.

2.3.2.5 Topologia BCube

A topologia *BCube*, Figura 3(e), é uma arquitetura centralizada no servidor construída em uma estrutura com dois níveis. Cada servidor é equipado com um pequeno número de interfaces que permitem estabelecer múltiplas conexões. Normalmente uma conexão com o *switch* de acesso, que conecta todos os servidores da célula e outra conexão com um *switch* de agregação. Desta forma, consegue aumentar a largura de banda agregando enlaces, além de melhorar a tolerância à falha e executar balanceamento de carga (NOORMOHAMMADPOUR; RAGHAVENDRA, 2017). O roteamento é realizado pelo servidor que precisa alterar a pilha *Transmission Control Protocol* (TCP). Uma alteração desse porte pode causar uma sobrecarga no uso da *CPU* em altas taxas de ocupação da rede (GUO et al., 2009). Nesta topologia os *switches* tem a função de estabelecer conexões com servidores dentro da célula. Nenhum *switch* é conectado diretamente a outro. As conexões com outras células são realizadas através de *switches* de agregação.

2.3.2.6 Topologia Dcell

A *Dcell*, representada na Figura 3(f), forma uma topologia hierárquica na qual uma célula com estrutura de nível mais alto agrupa várias células de estruturas de nível mais baixo. Cada equipamento é dotado de mais de uma placa de rede construindo uma topologia recursiva de células organizadas em vários níveis. A célula principal contém *N* servidores e um *switch* com a função de interligar os servidores dentro da mesma célula. É uma arquitetura escalável e apresenta elevada robustez estrutural. Ainda, é guiada por um algoritmo de roteamento customizado que leva em consideração os possíveis pontos de falhas ao procurar o caminho mais curto para o destino. Assim como na topologia *BCube*, a pilha TCP no servidor também precisa de alteração, uma vez que pode encaminhar a comunicação para outros servidores (GUO et al., 2008). Nesta topologia os *switches* têm a função de estabelecer conexões com servidores dentro da célula. Nenhum *switch* é conectado diretamente a outro. As conexões com outras células são realizadas através do servidor.

2.3.2.7 Considerações sobre as Topologias

É importante ressaltar que as topologias *Fat-tree* e *VL2* são semelhantes. São constituídas de três camadas em uma topologia *Clos*. A diferença entre elas é que a topologia *Fat-tree* separa a camada de borda em Agrupamentos (PODs), enquanto a topologia *VL2* agrega um conjunto de *switches* de borda conectando-os a 2 *switches* na camada de agregação. Com relação ao roteamento, as topologias *Fat-tree*, *VL2* e *Leaf-Spine* são orientadas por algoritmos de roteamento que executam *Equal-Cost Multi-Path Routing* (ECMP), uma situação na qual o fluxo pode ser balanceado entre vários caminhos disponíveis com o mesmo custo. Outras topologias como *BCube* e *Dcell* necessitam de alterações na pilha TCP dos servidores. A topologia *JellyFish* apresenta problemas de roteamento em função das constantes trocas de caminhos realizados pela conexões randômicas entre os *switches* (FILIPOSKA; JUIZ, 2014).

Sobretudo, é importante ressaltar que congestionamentos ocorrem em todas as topologias. De acordo com o tráfego ativo na rede, congestionamentos podem ser percebidos em enlaces com um único salto ou caminhos compartilhados. Assim, uma representação canônica para estudar congestionamentos independente da topologia do DC é a utilização de conexões em formato de alteres (também conhecida como topologia *Dumbbel*). Em suma, dois *switches* interligam grupos de servidores, formando o gargalo de comunicação entre os *switches*. Nos capítulos subsequentes, para caracterizar a comunicação TCP do HMR, sobretudo estudando o congestionamento, o presente trabalho utiliza a representação canônica oferecida pela topologia *Dumbbel*.

2.3.3 Virtualização de DCs

A virtualização de recursos computacionais é utilizada para representar e compartilhar recursos físicos. Podem ser virtualizados recursos como servidores, unidades de armazenamento, dispositivos do núcleo da rede como *switches* e roteadores, dispositivos finais, e todos os recursos subjacentes.

2.3.3.1 MVs e Contêineres

Alguns exemplos de virtualizadores utilizados em DCs são: VMware ESXi (VMWARE-ESXI, 2018), KVM (KVM, 2018), Citrix Xen (XEN, 2018) e Microsoft Hyper-V (MICROSOFT, 2018), etc. Com máquinas virtuais é possível utilizar em um mesmo servidor recursos que utilizam diferentes SOs. Em suma, as Máquina Virtuals (MVs) são ambientes gerenciados por *software* que simulam ambientes controlados por *hardware*. A escalabilidade em ambientes virtuais é obtida utilizando recursos automatizados de balanceamento de carga para distribuir a carga de trabalho entre os diferentes servidores que hospedam as MVs.

Uma tecnologia alternativa consiste no uso de contêiner, permitindo que um servidor, dotado de um único sistema operacional, execute várias aplicações isoladas, enquanto que uma MV pode executar em um mesmo nó computacional diversos sistema operacionals (SOs). A tecnologia de contêineres foi introduzida inicialmente em GNU/Linux com a finalidade de prover segurança, sendo que a escalabilidade da carga de trabalho é realizada sob demanda de acordo com os serviços básicos que suas funções hospedadas exigem (ZHANG; LU; PANDA, 2016; MAHJOUB et al., 2011).

A virtualização separa as aplicações dos recursos físicos permitindo que cada recurso virtualizado possua um ambiente confinado, no qual pode executar aplicações não confiáveis, assim, compartilhando recursos físicos como, por exemplo a placa de rede e outros recursos de processamento e armazenamento, bem como recursos do SO através de técnicas de isolamento. O hipervisor controla os recursos garantido acesso personalizado para cada estrutura aos recursos de rede, como provisionamento de largura de banda, endereçamento, versão de protocolo, roteamento e encaminhamento de forma logicamente isolada, visando proporcionar um alto nível de segurança (PRIMET; ANHALT; KOSLOVSKI, 2009). É importante ressaltar que as operações de configuração, posicionamento sobre os recursos físicos e manutenção do ambiente são realizadas pelo provedor.

2.3.3.2 Redes Virtuais

Uma *Virtual Local Network* (VLAN) é um exemplo de virtualização na qual uma rede *Ethernet* de grandes proporções é dividida em várias sub-redes (ZHOU; MA, 2016). Para diferenciar o tráfego entre as VLANs foi introduzido um campo no cabeçalho da camada de rede que armazena o número, ou *tag*, da VLAN a que o fluxo pertence (MCKEOWN et al., 2008).

Outra tecnologia muito utilizada é a *Virtual Private Network* (VPN) que consiste em uma rede privada criada sobre uma estrutura de rede compartilhada. Para estabelecer uma conexão VPN é necessário configurar um serviço que pode ser acessado por clientes mediante autenticação. Após a conexão entre o cliente e o serviço ser estabelecida, a comunicação entre eles pode ser realizada de forma segura (LUO et al., 2013).

Redes virtuais são abstrações de diferentes redes executadas sob o mesmo meio físico subjacente. A criação de redes virtuais tem sido favorecida pela tecnologia de virtualização de funções de rede, *Network Function Virtualization* (NFV), que proporciona flexibilidade na oferta de serviços de rede uma vez que desacopla sua implementação da rede física subjacente (HAN et al., 2015). Outra tecnologia, a *Software Defined Network* (SDN) é uma arquitetura que separa o plano de controle do plano de dados. O plano de controle fica hospedado em um controlador que tem uma visão geral, ao invés de ser replicado em cada *switch* da topologia. Quando um *switch* recebe um pacote e não tem uma indicação de qual porta deve encaminhá-lo, solicita ao controlador essa informação. Dessa forma, um *switch* pode manipular o fluxo de dados de diferentes redes. Um controlador pode gerenciar a tabela de encaminhamento de várias redes simultaneamente (GALLARDO; BAYNAT; BEGIN, 2016). Para obter essa flexibilidade utiliza o *OpenFlow* (MCKEOWN et al., 2008), um protocolo aberto que tem a finalidade de programar a tabela de encaminhamento de fluxos do *switch*, independente do modelo e fornecedor. É amplamente utilizado em redes SDN, permitindo ao administrador da rede controle sobre as rotas para as quais os diferentes fluxos podem ser encaminhados. O padrão *OpenFlow* é constituído de basicamente três partes: *(i)* a tabela de fluxos, na qual cada fluxo é associado a uma interface; *(ii)* um controlador que se comunica com o switch através de um canal seguro trocando informações sobre o plano de controle dos pacotes e *(iii)* o protocolo que garante um padrão aberto de comunicação entre o controlador e o *switch*. O controlador tem uma visão centralizada da topologia da rede.

Um *switch* virtual permite conectar elementos de rede, utilizando recursos de *software*, da mesma forma como podem serem conectados em *hardware*. Por exemplo, em hipervisores de máquinas virtuais, para conectar uma MV ao *switch* virtual deve ser selecionado o adaptador da rede que será associado à porta virtual. Ainda,

um adaptador de rede em modo *bridge* permite associar a MV ao adaptador Ethernet físico do servidor onde o virtualizador está executando, da mesma forma que é possível interconectar várias MVs executando no mesmo servidor. Assim, as interfaces virtuais não são conectadas a nenhuma conexão externa (VSWITCH, 2018).

Dentre as opções para implementação de *switches* virtuais, o *Open vSwitch* é comumente utilizado em arquiteturas *OpenStack* e *CloudStack*. Em suma, o *Open vSwitch* atua em conjunto com outros hipervisores e contêineres como Xen (XEN, 2018), KVM (KVM, 2018) e Docker (DOCKER, 2018), utilizando o protocolo *OpenFlow* para programar o fluxo de dados dinamicamente configurado. Diferente de outros *switches* virtuais, o *Open vSwitch* implementa *cache* de fluxos para não esgotar recursos do hipervisor. O encaminhamento de pacotes é constituído de dois componentes principais: o primeiro componente é o *ovs-vswitchd* que atua no espaço do usuário e é o mesmo para diferentes SOs; o segundo componente é o *datapath kernel* que atua no núcleo do sistema operacional e por questões de desempenho é específico para o SO do servidor (PFAFF et al., 2015).

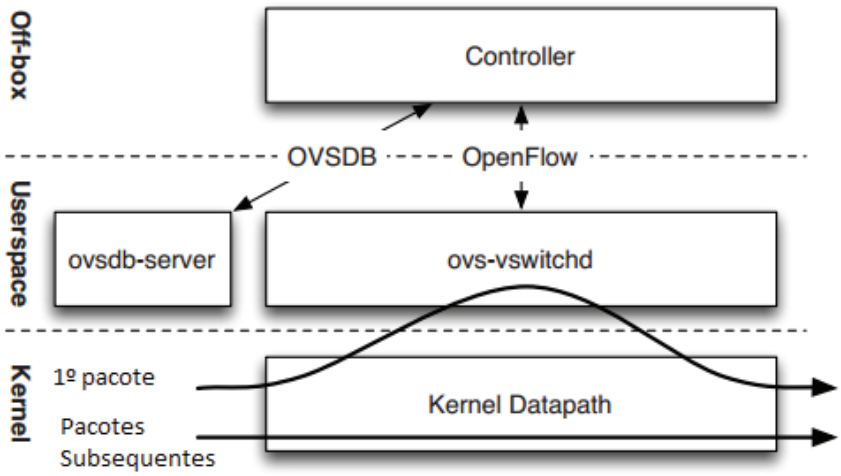


Figura 4 – Arquitetura do Open vSwitch Adaptado de (PFAFF et al., 2015).

Conforme representado na Figura 4, em um *Open vSwitch*, o primeiro pacote de um fluxo é encaminhado para o componente *ovs-vswitchd* no espaço do usuário que através do *Openflow* se comunica com o controlador e estabelece o caminho que esse fluxo deve seguir. Os pacotes seguintes são encaminhados orientados pela informação presente em *cache* no espaço do usuário (PFAFF et al., 2015).

Com a virtualização, o tráfego de diferentes aplicações é monitorado pelo hipervisor, ou seja, filtros podem ser aplicados baseados em classes de fluxos para definir as rotas, bem como as prioridades de encaminhamento. Com a separação do plano de controle do plano de dados, o dispositivo pode encaminhar o pacote recebido

para a rota estabelecida sem executar o processamento do algoritmo para a definição da rota. Tal processamento é transferido para o controlador que tem uma visualização da rede como um todo e consegue visualizar todos os possíveis caminhos bem como o estado dos enlaces. A virtualização do controle de congestionamento, detalhada no Capítulo 3, pode ser implementada no hipervisor ou diretamente no *switch* virtual.

2.4 CONTROLE DE CONGESTIONAMENTO

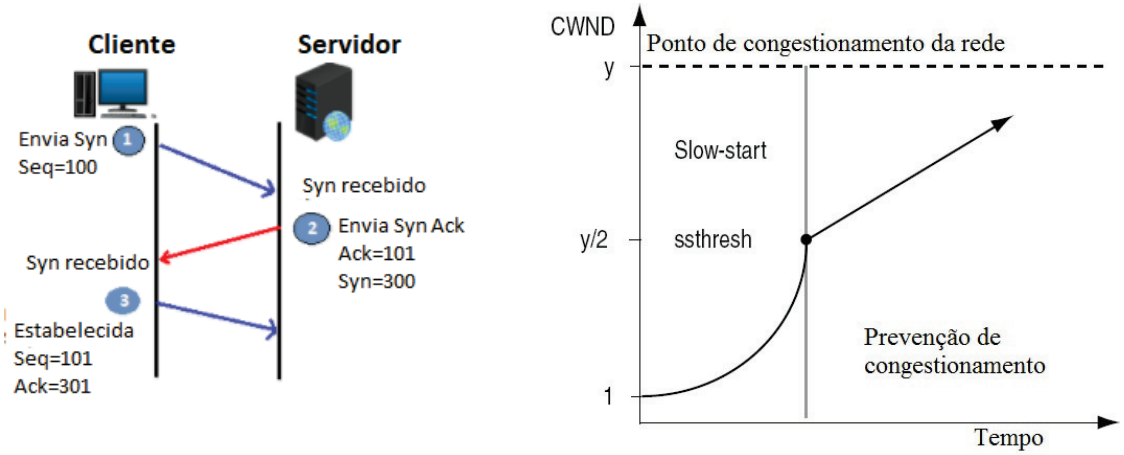
Independente da topologia descrita na Seção 2.3 e tecnologias de virtualização apresentadas na Seção 2.3.3 aplicados no DC, a ocorrência de congestionamento na comunicação é inevitável (NOORMOHAMMADPOUR; RAGHAVENDRA, 2017). Esta seção revisa as propostas implementadas para contornar o cenário de congestionamento. Quando muitos fluxos compartilham a rede ocorre uma competição pelo uso do meio de transmissão. Os fluxos TCP enviam um volume de dados e recebem uma confirmação *Acknowledgments* (ACK) do recebimento destes dados pelo destinatário.

Os dispositivos finais, responsáveis por iniciar e finalizar a comunicação, percebem a ocorrência de congestionamento pelo recebimento de ACKs duplicados, ou quando um temporizador é finalizado. Na Seção 2.4.1 é abordado como é realizada a conexão, e como o TCP identifica o congestionamento e trata o ajuste do tamanho da janela de congestionamento. Novas abordagens que procuram evitar a perda de pacotes são tratadas na Seção 2.4.2, no qual o controle de congestionamento é identificado pelo núcleo da rede e notifica as extremidades para agir conforme o volume de congestionamento ajudando a rede a drenar e ajustar o fluxo.

2.4.1 Controle de Congestionamento nas Extremidades da Rede

O TCP é um protocolo orientado a conexão que garante a entrega dos dados na ordem em que foi enviado. É implementado na camada 4 do modelo *Open System Interconnection* (OSI) ou TCP/IP (KUROSE; ROSS, 2006). Utiliza um mecanismo de troca de mensagens conhecido como *Triple Handshake* ou traduzindo, aperto de mão triplo, no qual estabelece a conexão negociando o tamanho da janela de dados e a sequência de envio conforme representado na Figura 5(a). Inicialmente, no passo 1, o cliente envia um pacote *SYN* com um número de sequência solicitando o estabelecimento de uma conexão TCP. O destinatário por sua vez, no passo 2, retorna um pacote *SYN-ACK* confirmando a sequência recebida e solicitando a próxima sequência. No passo 3, o cliente ao receber a confirmação, notifica o destinatário que recebeu a confirmação e começa a enviar os dados negociados na abertura da conexão. Quando um pacote é enviado, um temporizador é acionado para aguardar uma resposta do destinatário. Se o temporizador esgotar antes do recebimento de

um *Acknowledgments* (ACK), que confirma o recebimento do pacote pelo destinatário, a sequência de dados é reenviada. O congestionamento é identificado pelo recebimento de ACKs duplicados ou pela perda de pacotes. Quando isso acontece o TCP reduz o tamanho da janela de dados que está enviando para drenar o congestionamento da rede. O destinatário precisa ordenar os pacotes recebidos antes de entregar os dados para a aplicação.



(a) *Triple HandShake.* (b) *Slow Start.*

Figura 5 – Operações do TCP.

O *Slow Start*, representado na Figura 5(b), é um algoritmo que foi inicialmente proposto por (JACOBSON, 1988) e parte do princípio da conservação de pacotes. Seu objetivo é preencher o canal de comunicação sem causar congestionamento, colocando o sistema em seu estado inicial e estável. Para tal fim, foi introduzida uma nova janela, a *Congestion Window* (CWND), configurando seu valor inicial para um *Maximum Segment Size* (MSS) quando uma nova conexão TCP é iniciada. O CWND é uma variável do TCP que limita a quantidade de dados que o emissor pode enviar sem receber um pacote ACK, enquanto que o *Receiver Window* (RWND) é uma variável de controle do TCP que limita a quantidade de dados que o receptor pode manipular. As duas variáveis auxiliam o TCP no controle do fluxo de dados e definem o tamanho da janela de dados da transmissão. Ao receber o ACK deste primeiro segmento, o tamanho da janela é incrementado para dois segmentos. Logo após o remetente receber a confirmação destes dois pacotes, o algoritmo altera o tamanho do CWND para quatro segmentos, dobrando a capacidade de crescimento da janela a cada *Round Trip Time* (RTT), ou seja, um crescimento exponencial.

Quando ocorre um congestionamento, o algoritmo de controle de congestionamento é utilizado para drenar o fluxo congestionado. Para tal tarefa ele invoca o algoritmo de partida lenta, *Slow Start*, que inicia a retomada do fluxo a partir de um

MSS. Cada conexão mantém duas variáveis:*(i)* o CWND e *(ii) ssthresh* que representa o limite do algoritmo *Slow Start*. Se o CWND for menor ou igual ao *ssthresh* o TCP está operando em modo de *Slow Start* do contrário está operando com o algoritmo de prevenção de congestionamento o *congestion avoidance*.

Dentre as várias versões do TCP pode se destacar a versão *Reno* que para agilizar a retomada do tamanho da janela utiliza o recurso de retransmissão rápida com o algoritmo *Fast Retransmit*. Quando 3 ACK duplicados são identificados, o protocolo considera um indicativo de atividade na rede (e não de um colapso, pois o temporizador não foi esgotado), ou seja, não é necessário reiniciar a janela com o valor de 1 MSS. A recuperação rápida ocorre através da divisão da janela ao meio, somando a ela 3 MSS. Foi implementado inicialmente na versão do TCP 4.3BSD Tahoe (FLOYD; HENDERSON; GURTOV, 2004).

Por sua vez, o *Fast Recovery*, ou recuperação rápida, é um algoritmo que é invocado logo após a execução do algoritmo *Fast Retransmit* e tem a missão de recuperar rapidamente a janela de dados sem executar o *Slow Start*. Surgiu na implementação da versão do TCP 4.3BSD Reno (STEVENS, 1997).

Por sua vez, a versão *Cubic* (HA; RHEE; XU, 2008), padrão das distribuições GNU/Linux a partir da versão 2.6.19, é uma melhoria da versão *Bic* (HUA; GONG, 2012) e não depende de ACK para ajustar a janela de congestionamento. O tamanho é ajustado executando uma função cúbica sobre o valor da janela no momento em que ocorreu o último evento de congestionamento.

Por fim, independente da versão utilizada, é importante ressaltar que nos algoritmos elencados, a tomada de decisão ocorre sem o auxílio dos equipamentos da rede. Ou seja, o algoritmo infere as informações de acordo com as indicações recebidas: RTT, 3 ACKs recebidos e *timeout*.

2.4.2 Controle de Congestionamento com Auxílio do Núcleo

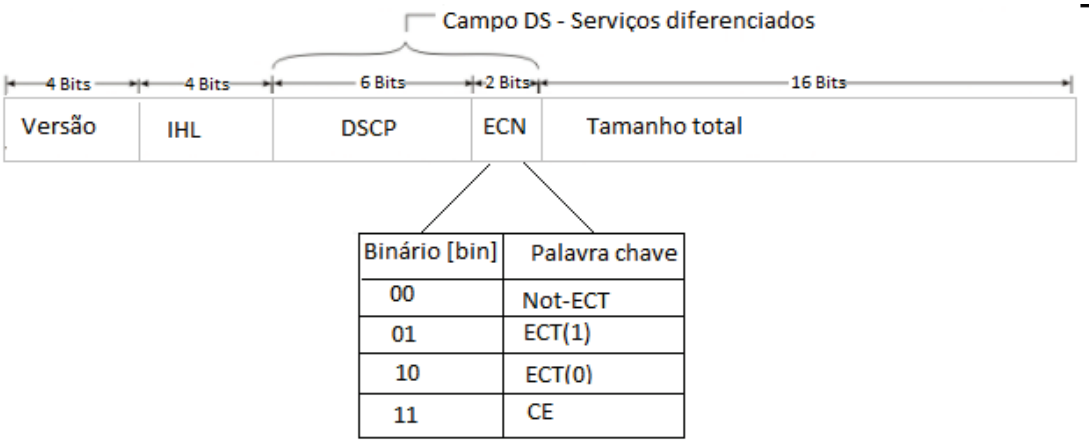
Redes de DC são projetadas visando alta vazão e baixa latência (AL-FARES; LOUKISSAS; VAHDAT, 2008). Entretanto, os *switches* comumente usados possuem arquitetura baseada em memória compartilhada. Assim, quando múltiplos fluxos convergem para a mesma interface caracterizando o fenômeno denominado *(incast)* (WU et al., 2013), o espaço de armazenamento em *buffer* pode ser totalmente consumido, ocasionando o descarte de pacotes (ALIZADEH et al., 2010). Como DCs de nuvens Infraestruturas como Serviço (IaaS) hospedam aplicações com padrões distintos de comunicação, àquelas baseadas em particionamento, processamento e agregação de informações trafegam dados de controle sensíveis à latência, bem como tráfego de fluxos para sincronização de massas de dados. Ou seja, os fluxos sensíveis à latência

concorrem com o tráfego de *background*, constituído por pequenos e grandes fluxos. Analisando a ocupação dos *buffers* em *switches* e servidores, fluxos maiores que 1 MB possuem baixa multiplexação e consomem grande parcela do espaço disponível (ALIZADEH et al., 2010). Consequentemente, induzem a formação de filas e o aumento da latência para os demais fluxos.

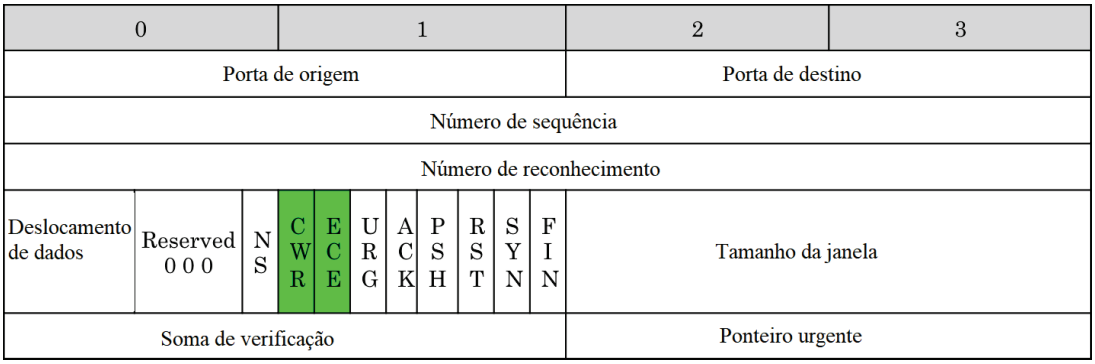
Topologias com vários níveis como a *Fat-tree* e a VL2, oferecem múltiplos caminhos para o mesmo destino, com isso podem fazer o balanceamento de carga, minimizando o impacto da formação de filas ao direcionar o fluxo de pacotes para outras interfaces de saída para caminhos alternativos menos congestionados, liberando o *buffer* para outros pacotes que estão chegando no *switch*. Enquanto que topologias com roteamento realizado pelo servidor, como o caso da *Bcube* e *Dcell*, o pacote permanece na fila, até que todos os que o sucederam sejam encaminhados numa dinâmica de manipulação de fila em que o primeiro que entra é o primeiro que sai *First In First Out* (FIFO). Em alguns casos o atraso acrescentado na espera pelo processamento na interface contribui para a saturação do *buffer*, consequentemente descartando pacotes, além de influenciar o *timeout* e disparar retransmissões desnecessárias.

Por ser baseado em *feedback* binário (CHIU; JAIN, 1989), o TCP apenas altera seu modo de transmissão na ocorrência de perda de segmentos ou recebimento de ACKs duplicados. Na ausência de eventos de perdas, a janela de transmissão é aumentada, enquanto na ocorrência, a mesma é diminuída (HA; RHEE; XU, 2008; JACOBSON, 1988). Nativamente, conforme discutido na Seção 2.4, o congestionamento é percebido quando um temporizador é finalizado ou quando um conjunto de ACKs duplicados são recebidos pelo emissor. Embora eficiente nos cenários originalmente propostos, os algoritmos com controle de congestionamento baseada em informações trocadas pelos equipamentos nas extremidades da rede são inaptos para detectar e controlar o cenário heterogêneo, comum em DCs (ALIZADEH et al., 2010) com múltiplos inquilinos hospedando aplicações com técnicas diferenciadas de inferência e controle do congestionamento.

Assim, a técnica de *Explicit Congestion Notification* (ECN), uma extensão da pilha TCP/IP (KUZMANOVIC et al., 2009), alterou o *feedback* recebido pelo emissor TCP, ou seja, a perda de pacotes é eventualmente substituída por um aviso da possível ocorrência de congestionamento. De posse do aviso, o emissor pode preventivamente reduzir o volume de dados trafegados, atenuando a formação de filas nos dispositivos intermediários. Quanto aos protocolos, a implementação desse mecanismo foi realizada através da introdução de 2 *bits* na camada de rede: *ECN-Capable Transport* (ECT) e *Congestion Experienced* (ECE), conforme representado na Figura 6(a). O primeiro *bit*, indica que o equipamento é capaz de transportar a notificação de con-



(a) ECN - Camada de Rede



(b) ECN - Camada de Transporte

Figura 6 – Informações sobre ECN nos cabeçalhos das camadas 3 e 4.

gestionamento, enquanto o segundo sinaliza que uma situação de congestionamento está acontecendo. Na camada de transporte, quando o *bit* ECE é percebido, o destinatário acrescenta a mesma informação ao segmento de confirmação de recebimento, conforme exemplificado pela Figura 6(b). Ao receber a notificação, o emissor reduz a janela de congestionamento para evitar a perda de segmentos e informa sua ação ao receptor, através do *bit Congestion Window Reduced* (CWR).

A marcação de pacotes que passam por situações de congestionamento é realizada pelos *switches* intermediários. Para tal, um monitoramento constante das filas de encaminhamento é realizado, disparando ações de acordo com parâmetros previamente informados. Nesse contexto, configurações de RED (WU et al., 2012) podem ser aplicadas, especificando: *(i)* a largura de banda máxima do enlace (bytes/s); *(ii)* tamanho mínimo, máximo e instantâneo (para atender rajadas) da fila (bytes); *(iii)* tamanho médio dos pacotes; *(iv)* tolerância para o tamanho instantâneo da fila em pacotes; e *(v)* probabilidade de descarte (de 0,0 a 1,0). Quando o pacote é recebido pelo equipamento e se mantêm abaixo do mínimo especificado, não ocorre marcação.

Entretanto, pacotes que estão entre os limiares mínimo e máximo são marcados de acordo com a probabilidade informada. Por fim, os pacotes acima do limiar máximo são descartados. Dentre as variantes do TCP com interpretação do ECN, o *Data Center TCP* (DCTCP) foi desenvolvido visando tolerância a rajadas, baixa latência e alta vazão, utilizando *switches* de prateleira com *buffer* raso, oferecendo um aumento na vazão quando comparado às variantes tradicionais (ALIZADEH et al., 2010).

2.4.3 Limitações do controle de congestionamento em DCs

As Máquina Virtuals (MVs) hospedadas em DCs são heterogêneas quanto ao algoritmo de controle de congestionamento que utilizam, da mesma forma hospedam aplicações que apresentam diferentes tamanhos de fluxos. Fluxos de pacotes com tamanho significativo por exemplo, um transmissão de vídeo, compartilham o *buffer do switch* com fluxos pequenos de uma conexão *Secure Shell* (SSH). Estes últimos são sensíveis a latência, enquanto que os primeiros são caracterizados por necessitarem de uma grande vazão. Essas diferentes cargas fazem com que fluxos sensíveis a latência permaneçam na fila enquanto aguardam que os pacotes sejam processados e encaminhados, aumentando significativamente a latência percebida pelo usuário (ALIZADEH et al., 2010).

É fato que o mecanismo de ECN permitiu um melhor compartilhamento dos recursos de comunicação em DCs (ALIZADEH; JAVANMARD; PRABHAKAR, 2011). Todavia, a efetiva aplicação requer uniformidade dos algoritmos, ou seja, os servidores devem compreender o significado das marcações efetuadas pelo núcleo da rede. Esse requisito não é observado em DCs de nuvens IaaS, sendo que cada inquilino pode executar versões distintas de algoritmos. Ainda, surpreendentemente, diversas versões de SOs não possuem suporte ativado por padrão para ECN (KüHLEWIND; NEUNER; TRAMMELL, 2013). Ou seja, embora o DC implemente mecanismos para otimizar o tráfego, as configurações realizadas nas MVs dos usuários podem estar em desacordo com o cenário ideal.

Múltiplos inquilinos no DC com um conjunto de aplicações com necessidades tecnológicas diferentes, algumas desatualizadas (CRONKITE-RATCLIFF et al., 2016), criam uma diversidade de algoritmos de controle na rede do DC. Diferentes padrões de tráfego e algoritmos para controle de congestionamento com comportamento distintos penalizam a justiça e equidade na rede (BENSON; AKELLA; MALTZ, 2010). Versões mais agressivas do algoritmo de controle de congestionamento tendem a monopolizar o fluxo causando uma insatisfação em clientes com algoritmos menos agressivos, comprometendo inclusive acordos de Acordo de Nível de Serviço (SLA) que deixam de ser atendidos ocasionando multas e perdas de receitas. Neste cenário, o provedor não pode interferir na pilha TCP das máquinas do cliente porque as aplicações têm

uma forte dependência de bibliotecas do SO (além de ferir princípios de segurança e isolamento).

Em configurações ECN com gerenciamento de uma única fila, ocorre uma extensiva negociação entre alta vazão, baixa latência e equidade do compartilhamento da rede. O limite de marcação pode ser atingido mediante a soma de todos os fluxos ocupando a fila. Ao aplicar um gerenciamento baseado em uma única fila, pode se obter uma maior vazão e baixa latência para alguns fluxos, mas viola o princípio de equidade no compartilhamento da rede. Para amenizar este problema (BAI et al., 2016) propõe uma forma de gerenciamento de filas baseado em múltiplas filas, com limites de marcação independentes. O limite de cada fila é configurado de forma dinâmica, baseado na ponderação da taxa de ocupação. Essa alteração implica em alteração na estrutura de gerenciamento de filas no *switch*.

2.5 CONSIDERAÇÕES PARCIAIS

O grande volume de dados gerados em decorrência da execução de aplicações modernas aumentou significativamente a complexidade de processamento, bem como de armazenamento. Essa dificuldade motiva o surgimento de sistemas de armazenamento distribuídos, e processamento em paralelo. O *MapReduce* torna-se uma proposta interessante uma vez que esse conceito de programação se beneficia da replicação do armazenamento para posicionar o processamento mais próximo possível do local dos dados, minimizando as exigências de comunicação na rede. Dentre as várias ferramentas do arcabouço *Hadoop*, o HMR mapeia as entradas em um conjunto de [par,valor] e distribui o processamento entre os *servidores* executando em paralelo.

Ainda, a rapidez da evolução das tecnologias torna difícil para as empresas manterem atualizados os recursos necessários para o andamento de seus negócios. Contratar fornecedores que realizem esse trabalho se tornou algo comum. Dessa forma, as empresas se concentram nas atividades que dizem respeito ao negócio, e o aparato tecnológico é terceirizado. Do ponto de vista do cliente não, existe a preocupação em qual tecnologia o provedor de serviço utiliza. O mais importante é o cumprimento do SLA. Do ponto de vista do provedor, ele precisa lidar com uma diversidade de tecnologias que utilizam o mesmo meio físico. Por exemplo, comunicações entre aplicações que utilizam SO com uma pilha TCP otimizada coexistem com outras aplicações onde os recursos do TCP não são otimizados. As diferentes versões do algoritmo TCP têm mecanismos diferentes para identificar e se recuperar quando ocorrem eventos de congestionamento e perdas de pacotes.

Alguns melhoramentos têm sido propostos para atender a crescente demanda nestes ambientes diversificados que se tornaram os DCs prestadores de serviços em

nuvem. O ECN é um mecanismo que permite ao núcleo da rede identificar a formação de congestionamento, antes que perdas de pacotes aconteçam. Essas perdas têm como consequência a necessidade de reenvio destes pacotes, contribuindo para uma diminuição da vazão. As redes de DCs trabalham com o princípio de obter alta vazão e baixa latência. Os protocolos da pilha TCP têm se aperfeiçoado, no sentido de se recuperar mais rapidamente de situações que diminuem o tamanho da janela, como o algoritmo *Fast Retransmit* e *Fast Recovery* no qual a janela não é fechada completamente para iniciar a retomada da vazão.

Esses recursos associados a possibilidade de virtualizar o controle da rede, tem oferecido uma opção para os provedores de nuvens computacionais de manter múltiplos clientes com diferentes tipos de aplicações coexistindo de forma justa e equitativa conforme discutido no Capítulo 3.

3 VIRTUALIZAÇÃO DO CONTROLE DE CONGESTIONAMENTO (VCC)

Neste capítulo são discutidos avanços no controle de congestionamento em *data center*s (DCs) virtualizados. Na Seção 3.1 são abordados o conceito de virtualização e as tecnologias que fazem parte do processo de tradução dos algoritmos de controle de congestionamento. Na Seção 3.2, são descritos a arquitetura e as técnicas utilizadas para implementação da Virtualização do Controle de Congestionamento (VCC), assim como são abordados as ferramentas utilizadas para compor o cenário em estudo. Na sequência, na Seção 3.3, são descritos alguns trabalhos de autores que estudam o tema e os avanços que fizeram em suas pesquisas. Por fim a Seção 3.4 apresenta as considerações finais do capítulo.

3.1 CONCEITOS E DEFINIÇÕES

A virtualização do controle de congestionamento consiste em criar uma camada para tradução do protocolo *Transmission Control Protocol* (TCP) utilizado pelas Máquina Virtuals (MVs) em uma versão otimizada e reconhecida pelo *data center* (DC) (HE et al., 2016; CRONKITE-RATCLIFF et al., 2016). Dessa forma, independente da versão utilizada pelo cliente, a comunicação ocorre com a versão selecionada pelo provedor. É importante ressaltar que nenhuma alteração deve ser efetuada nos hospedeiros finais (MVs ou sistema operacionals (SOs)) para não afetar premissas de segurança. As informações necessárias para que uma máquina com algoritmo TCP que não utiliza *Explicit Congestion Notification* (ECN) possa se comunicar com outra máquina que utilize a notificação explícita de congestionamento são introduzidas na conexão TCP pelo hipervisor. Algumas técnicas de tradução apontadas por (CRONKITE-RATCLIFF et al., 2016) são:

- Escrever na memória do convidado: nesta técnica o hipervisor tem uma visão completa do estado da memória da MV do inquilino e pode escrever nela quando necessário;
- Ler da memória do convidado: semelhante a técnica descrita acima, o hipervisor pode monitorar o estado da memória das MVs dos inquilinos, porém não tem permissão para escrita;
- Dividir a conexão: separar a conexão TCP em várias sub-conexões as quais podem ser encaminhadas dentro do DC utilizando diferentes caminhos e algoritmos de controle de congestionamento;

- Armazenamento temporário de pacotes: o hipervisor pode armazenar temporariamente pacotes em andamento e reenviá-los sem informar a MV do inquilino;
- Armazenamento temporário de *Acknowledgments*s (ACKs): o hipervisor armazena temporariamente ACKs ao invés de pacotes;
- Duplicar ACKs: o hipervisor duplica e reenvia um pacote ACK para forçar o inquilino a dividir sua janela de congestionamento;
- Estrangular a janela do receptor: o hipervisor decrementa a janela do receptor forçando o inquilino a diminuir o número de pacotes enviados; e
- Modificar o *Triple Hand Shake*: nesta técnica o hipervisor altera os parâmetros negociados durante o estabelecimento da conexão TCP inserindo informações, como ECN, em fluxos de MVs com TCP não otimizado.

Dentre as técnicas de tradução abordadas acima, a forma utilizada pelo presente trabalho, e pela literatura abordada, no qual ele se baseia (CRONKITE-RATCLIFF et al., 2016; HE et al., 2016) é a técnica de tradução e estrangulamento da janela do recebedor, auxiliada pela técnica de modificação do *Triple Hand Shake*. Como o controle é implementado no *datapath* do *switch* virtual (PFAFF et al., 2015) ou no hipervisor, conforme revisado na Seção 2.3.3, todo o tráfego pode ser monitorado. Estes equipamentos permitem que o administrador tenha controle sobre o algoritmo de controle de congestionamento sem alterar a pilha nas MVs dos inquilinos. Inclusive oferece uma granularidade de configuração que permite escolher o algoritmo em função do tipo de fluxo. Por exemplo, *Cubic* para tráfego interno e *Reno* para tráfego externo.

3.2 IMPLEMENTAÇÃO DO CONTROLE DE CONGESTIONAMENTO

Duas abordagens para implementação da VCC foram recentemente propostas. Inicialmente, o Administração Centralizada de Controle de Congestionamento (ACDC), proposto por (HE et al., 2016), implementa VCC em *switches* virtuais, obtendo uma fina granularidade no controle. Assim, algoritmos podem ser selecionados para diferentes tipos de fluxos, por exemplo, Cubic para fluxos externos ao DC e *Data Center TCP* (DCTCP) para fluxos internos. DCTCP é uma versão específica para atender as exigências de baixa latência e alta vazão em redes de DC (ALIZADEH et al., 2010). Por sua vez, (CRONKITE-RATCLIFF et al., 2016) propôs uma virtualização usando uma alteração no *kernel* do SO, apresentando uma prova de conceito utilizando o hipervisor proprietário VMware (VMWARE-ESXI, 2018) e MVs *Linux*.

Os inquilinos de MVs que utilizam algoritmos que percebem o congestionamento quando esgota um temporizador ou com recebimento de ACKs duplicados, dependem do virtualizador para complementar as informações de marcação ECN para

que o congestionamento seja percebido pelo núcleo da rede do DC. Dessa forma, dentro do DC é executado um algoritmo que usa o recurso de notificação explícita (ECN) em conjunto com um gerenciador de filas (como o *Random Early Detection* (RED), por exemplo). Assim, o hipervisor é responsável por traduzir o algoritmo da MV para a versão utilizada no DC, independente da configuração internamente realizada pelo cliente (MV legada ou TCP agressivo).

A Figura 7 apresenta a arquitetura genérica para virtualização do controle de congestionamento oriunda da combinação das propostas existentes. Quando uma aplicação legada estabelece uma conexão, o hipervisor ou *switch* virtual monitora a troca de pacotes e introduz as informações necessárias para que o tráfego não otimizado seja reconhecido pela rede como tráfego capaz de suportar ECN. Inicialmente, o TCP não otimizado envia um pacote solicitando a conexão (1), que é interceptado (2) para acrescentar as informações de suporte ECN. O destinatário responde confirmando o estabelecimento (3, *Congestion Experienced* (ECE)). Novamente, o pacote é interceptado para notificar o remetente com um reconhecimento da sincronização (4). É importante observar que a confirmação enviada para o emissor sem otimização do TCP não possui as informações sobre ECN, que foram removidas pelo virtualizador. E envio efetivo dos dados inicia (5), sendo interceptados para acréscimo do bit *ECN-Capable Transport* (ECT), informando que esse fluxo é capaz de transportar informação de congestionamento (6). Posteriormente, o destinatário reconhece o pacote recebido (7). O remetente recebe do hipervisor o reconhecimento de pacote (8). No caso de possível ocorrência de congestionamento na rede do DC, o bit ECE é ativado (9). Quando ocorre congestionamento, o emissor com algoritmo não otimizado é obrigado a reduzir o envio de dados através do estrangulamento da janela de recepção (10), realizado pelo hipervisor. Por fim, o remetente continua o envio de dados (11) para o hipervisor que transfere os dados e o tamanho da janela ajustada para o destinatário (12).

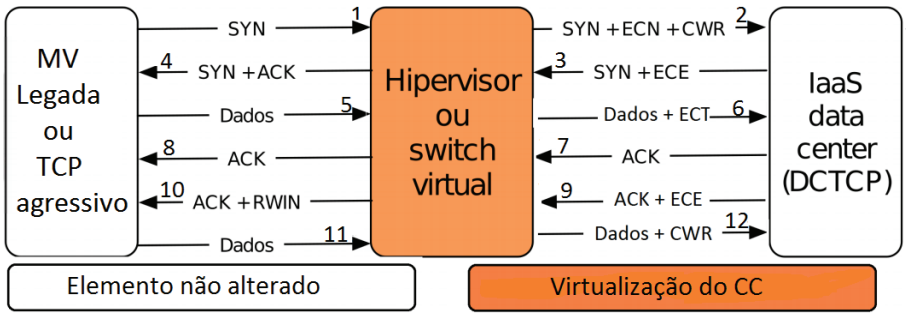


Figura 7 – Arquitetura para virtualização do controle de congestionamento, adaptado de (CRONKITE-RATCLIFF et al., 2016).

Para induzir a desaceleração de transmissão do hospedeiro sem aplicar uma

técnica intrusiva, o controle de congestionamento é aplicado sobre a janela de recepção, *Receiver Window* (RWND). Assim, as informações internamente aferidas pelo algoritmo na MV, sobre a janela de congestionamento, CWND, permanecem inalteradas. Nativamente, o TCP verifica $min(cwnd, rwnd)$ para identificar o volume de dados que podem ser trafegados em um determinado instante. No ambiente virtualizado, o valor de RWND é alterado pelo hipervisor para representar o correto valor de CWND calculado pelo algoritmo do virtualizador, baseado em ECN. Por exemplo, uma MV utilizando o *Cubic* para controle de congestionamento, sem suporte à ECN, será convertida em um controle realizado por DCTCP.

A implementação do mecanismo ACDC (HE et al., 2016) foi desenvolvida utilizando hipervisor proprietário, não estando, portanto disponível à comunidade. No entanto, (CRONKITE-RATCLIFF et al., 2016) desenvolveram uma prova de conceito utilizando GNU/Linux sendo permitindo a utilização deste hipervisor para testes experimentais. Especificamente, VCC foi implementado aplicando uma alteração no núcleo do hipervisor para reconhecer ECN e traduzir o algoritmo com TCP não conforme (CRONKITE-RATCLIFF et al., 2016). Em suma, um conjunto de *buffers* intermediários são criados para cada conexão TCP, ou seja, o hipervisor monitora a comunicação das MVs. Os recursos do VCC e ECN são ativados diretamente através da manipulação dos parâmetros no *kernel* do SO utilizando o conjunto de comandos da ferramenta *sysctl*, conforme exemplificado pela Figura 8. No exemplo, um trecho de código Python é utilizado para ativar ECN para o HMR e ativar VCC. O trecho foi coletado do código utilizado na análise experimental, discutida no Capítulo 5. As configurações inicialmente informadas para um soquete TCP permanecem válidas até o término da comunicação.

```
    ..................
host.popen("sysctl -w net.ipv4.tcp_ecn=%d" %(ativarECNHadoop))
host.popen("sysctl -w net.ipv4.tcp_vtcp=%d" %(ativarVCC))
    ..................
```

Figura 8 – Trecho de código com o comando de ativação da VCC e do ECN.

O algoritmo da VCC, implementado experimentalmente no núcleo do sistema operacional Linux (CRONKITE-RATCLIFF et al., 2016), segue o padrão de execução em kernel, conforme representado na Figura 9 para tratar os ACKs recebidos. Inicialmente a ativação do VCC via configuração do *sysctl* é verificada, representado pelo condicional VCTP$== 1$. Se não está ativado segue o padrão convencional do TCP para tratar as mensagens ACKs recebidas. Posteriormente, o terceiro teste condicional do fluxograma verifica se: *(i)* o bit *ECE* está ativado no pacote recebido?; *(ii)* não é uma mensagem de sincronização, ou seja, nao é o início da conexão TCP?; *(iii)* o ECN

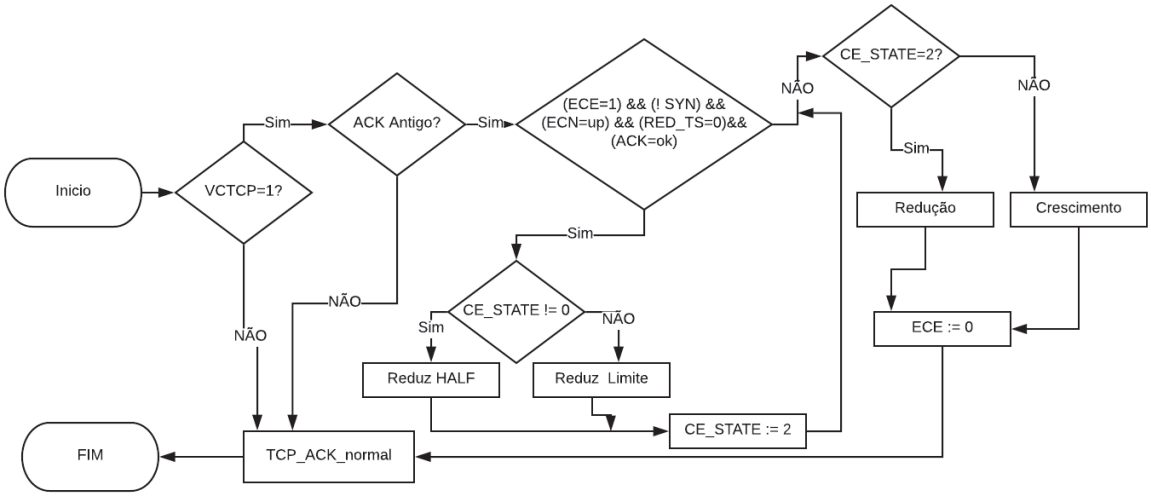


Figura 9 – Fluxograma de ativação do VCC guiado pelas configurações fornecidas via *Sysctl*.

está ativado no sistema operacional?; *(iv)* a janela não está reduzindo por uma execução anterior do algoritmo?; e *(v)* é realmente uma mensagem ACK? Sendo verdadeiro este teste, é inferido que o pacote carrega informação da ocorrência de congestionamento ao longo da rede, ou seja, a ação de redução da janela deve ser tomada, baseada no valor do bit *CE-STATE*. Se o valor for diferente de zero, a janela é divida pela metade. Do contrário a janela é fechada na proporção de pacotes marcados. Na sequência, a ação tomada é registrada ao alterar o bit *CE-STATE*.

Caso o valor do terceiro teste seja falso, o valor do bit *CE-STATE* é verificado para identificar se a decisão já foi previamente tomada. Deste ponto em diante, sendo o valor do bit *CE-STATE* diferente de 2, a janela se mantém em estado de crescimento. Caso contrário, entra em estado de redução. Na sequência é atribuído *zero* ao bit *ECE* e o TCP assume o fluxo normal. É importante ressaltar que a implementação descrita é uma prova de conceito, não otimizada para execução em *switches* virtualizados. Entretanto, os autores demonstraram que a sobrecarga computacional não impacta no desempenho final das comunicações, ou seja, a implementação pode ser utilizada para análise experimental controlada (CRONKITE-RATCLIFF et al., 2016).

3.3 TECNOLOGIAS E TRABALHOS RELACIONADOS

Os trabalhos relacionados compreendem estudos sobre notificação explícita de congestionamento, técnicas de virtualização do controle de congestionamento e estudos sobre otimização do tráfego de rede para HMR, além de outras tecnologias relacionadas ao ambiente de rede em DC.

Para facilitar a leitura, a apresentação dos trabalhos relacionados foi dividida

em três grupos distintos:*(i)* Na Seção 3.3.1 é apresentado os trabalhos que abordam a virtualização, gerenciamento de redes em DCs e tecnologias ECN e RED; *(ii)* Na Seção seguinte 3.3.2 é apresentado os trabalhos, cujos experimentos foram realizados utilizando a ferramenta HMR; *(iii)* E finalmente na Secão 3.3.3 são abordados os trabalhos, que não abordam os temas das seções anteriores, mas que serviram de base bibliográfica para a elaboração deste trabalho.

3.3.1 Virtualização e Gerenciamento de Redes em DCs

A Tabela 1 apresenta um resumo das técnicas utilizadas para gerenciamento de tráfego TCP em DCs.

Artigo	Contribuição	RED	ECN	Virtualização
(FLOYD, 1994)	Discutem a alteração do cabeçalho do pacote *IP* para permiter que o núcleo da rede possa transportar informação de congestionamento. Realizam testes mostrando as vantagens do ECN na predição de congestionamento, antes da ocorrência de perdas de pacotes.	Sim	Sim	Não
(KüHLEWIND; NEUNER; TRAMMELL, 2013)	Realizam uma análise do uso do ECN desde sua padronização em 2001. Mostram testes do uso deste mecanismos com versão *IPV6* do protocolo de endereçamento de rede. Ainda analisam outros recursos de controle de congestionamento do TCP como a *Selective Acknowledgments* (SACK).	Sim	Sim	Não
(CRONKITE-RATCLIFF et al., 2016)	Introduzem um *hipervisor* que traduz os algoritmos de controle de congestionamento em DC com múltiplos inquilinos, para um único algoritmo interno.	Sim	Sim	Sim
(HE et al., 2016)	Permitem ao administrador do DC ter controle sobre o algoritmo de controle de congestionamento, independente de qual os inquilinos utilizem. Permite ainda uma granularidade de algoritmos de acordo do fluxo, por exemplo, permite usar o algoritmo *Cubic* para fluxos locais, e outro algoritmo para fluxos com abrangência maior, como fluxos saindo ou chegando do meio externo à rede local.	Sim	Sim	Sim
(ALIZADEH et al., 2010)	Propõem o DCTCP, um protocolo TCP especifico para DC que faz uso do ECN, com a finalidade de obter alta vazão e baixa latência.	Sim	Sim	Não
(POPA et al., 2012)	Abordam os desafios existentes em redes virtualizadas para obter um mínimo da banda contratada enquanto mantém alta utilização e proporcionalidade na ocupação da rede.	Sim	Sim	Não
(KUZMANOVIC, 2005)	Realizam testes onde constatam que o ECN pode melhorar em 40% a vazão e consequentemente o tempo de resposta em aplicações *web*.	Sim	Sim	Não
(BAI et al., 2016)	Propõem o *MQ-ECN*, um recurso de gerenciamento de filas que estabelece e gerencia limites para várias filas simultaneamente.	Sim	Não	Sim

Tabela 1 – Trabalhos relacionados com virtualização e gerenciamento de redes em DCs.

Inicialmente, (FLOYD, 1994) discutiu o uso do mecanismo ECN no TCP. As simulações utilizando o algoritmo Reno e marcações guiadas por RED indicaram os benefícios deste recurso no controle de congestionamento, evitando perdas ao antecipar e prever situações prováveis de saturação da rede.

Por sua vez, (KüHLEWIND; NEUNER; TRAMMELL, 2013) abordou a evolução histórica do mecanismo ECN e sua padronização nos SOs, apresentando uma visão

de outras formas de prevenir e contornar congestionamento, como SACK. A justiça no compartilhamento de canais congestionados foi analisada em (HASEGAWA; MURATA, 2001). Os autores compararam variantes Reno e Vegas, aplicando tráfego de TCP e UDP como *background*. Os trabalhos indicaram a ausência de uma solução *de facto* para contornar a disparidade de compartilhamento quando múltiplos algoritmos para controle de congestionamento estão compartilhando e gerenciando gargalos. Ou seja, eles evidenciam o cenário potencialmente caótico que pode acontecer em DCs IaaS. Nesse cenário as aplicações que utilizam um algoritmo de controle de congestionamento mais agressivo conseguem obter o máximo de desempenho, no entanto, as aplicações com algoritmos desatualizados tendem a ser penalizados e não obterem largura de banda suficiente para execução com desempenho satisfatório, afetando acordo de SLA, consequentemente perda de receita para o provedor.

Consciente da diversidade de aplicações executando em um DC de nuvem com pilhas TCP/IP desatualizadas, os autores (CRONKITE-RATCLIFF et al., 2016; HE et al., 2016) propuseram a flexibilização na configuração, definindo o conceito de VCC. Estes trabalhos motivaram a realização da presente caracterização do HMR, propondo a utilização de múltiplos *switches* em duas topologias distintas: *Dumbbell* e *Fat-tree* . O estudo original contemplava uma topologia estrela com um único *switch*, analisando a aplicabilidade do VCC com cargas sintéticas que não representavam nenhuma aplicação em específico.

Ainda, o trabalho (ALIZADEH et al., 2010) identificou padrões de comunicação em DC que hospedam HMR: tráfego de consulta, tráfego de *background*, fluxos concorrentes e de diferentes tamanhos. A análise compreendeu o tráfego de 6000 servidores, coletando a formação de fila e pressão sobre *buffers*. De forma complementar, (WU et al., 2012) aborda o congestionamento causado por tráfego *incast* que ocorre quando múltiplos fluxos convergem para o mesmo receptor (gargalo e compartilhamento de filas).

Quanto ao gerenciamento de redes em DCs IaaS, (POPA et al., 2012) aborda a dura negociação ao estabelecer políticas de alocação de largura de banda visando:*(i)* garantir uma proporcionalidade de utilização da rede para os diferentes fluxos;*(ii)* conceder uma garantia mínima para o fluxo das MVs;*(iii)* evitar ociosidade e *(iv)* garantir alta ocupação da rede.

A opção de utilizar marcadores do núcleo da rede para inferir a ocorrência de congestionamento proporcionada pelo ECN foi abordada por (KUZMANOVIC, 2005), onde constatou um aumento de aproximadamente 40% na vazão de aplicações *web* comparada com a forma tradicional de percepção de congestionamento. Embora, conscientes das vantagens da predição antecipada de congestionamento no desempenho da rede, (BAI et al., 2016) expõem as dificuldades do ECN para garantir alta

vazão, baixa latência e um compartilhamento justo e equitativo dos recursos de rede. Para solucionar este problema propõem o *MQ-ECN*, um recurso de gerenciamento de filas que estabelece e gerencia limites para várias filas simultaneamente

3.3.2 Otimização e Gerenciamento de Aplicações HMR

Na Tabela 2 são apresentados os trabalhos relacionados, cujos experimentos, foram realizados utilizando a aplicação HMR.

Artigo	**Contribuição**	**Topologia**	**Nuvem *IaaS***	**RED**	**ECN**	**Virtualização**
(KOUBA; TOMANEK; KENCL, 2016)	Avaliam o impacto que diferentes topologias conhecidas e com diferentes cargas exercem em aplicações HMR.	Várias	Não	Não	Não	Não
(SOUZA et al., 2015)	Analisam o impacto de modelo de escalonamento computacional no desempenho de aplicações HMR.	Não informado	Não	Não	Não	Sim
(GUNARATHNE et al., 2010)	Propõem um *framework* para implementar aplicações HMR em ambiente de nuvem computacional.	Não informado	Sim	Não	Não	Não
(NEVES; ROSE; KATRINIS, 2015)	Propõem o *MRemu*, uma ferramenta que simula a comunicação de tarefas da aplicação HMR utilizando rastros de execução coletadas em *logs* de servidores reais.	Várias	Não	Não	Não	Não
(DEAN; GHEMAWAT, 2008)	Utilizam HMR para distribuir tarefas de computação em larga escala em um aglomerado de *cluster* construído sobre computadores convencionais.	Não informado	Não	Não	Não	Não
(RISTA DALVAN GRIEBLER, 2017)	Para melhorar a performance de aplicações HMR em redes virtualizadas, experimentam um conjuntos de *cluster* com instancias baseada em *container*.	Não informado	Sim	Sim	Não	Sim

Tabela 2 – Trabalhos relacionados com aplicação HMR.

A popularização de aplicações HMR no processamento de grandes volumes de dados e a necessidade de identificar o impacto que a topologia da rede pode proporcionar em aplicações desta natureza levou (KOUBA; TOMANEK; KENCL, 2016) a pesquisar as diferentes cargas de trabalho em diferentes topologias. Os autores concluíram que a escolha adequada da topologia pode melhorar significativamente a performance de aplicações HMR.

(SOUZA et al., 2015) utilizaram um *Virtual Private Server* (VPS) e *benchmarks* para analisar o desempenho de aplicações HMR utilizando diferentes modelos de escalonamento computacionais: local, geograficamente distribuído e centralizado. Os resultados obtidos são confrontados com servidores reais, visando apresentar um modelo alternativo de implementação de processamento de grandes volumes de dados tolerante a falhas, escalável e confiável.

(GUNARATHNE et al., 2010) analisaram os desafios inerentes a implementação de aplicações HMR em ambiente de nuvens computacionais, tais como: armaze-

namento de dados e metadados, comunicação consistente e confiável, escalabilidade, tolerância a falhas, entre outros. Ainda, discutiram alguns *frameworks* dos principais provedores de nuvens IaaS como, o *Amazon Elastic Map Reduce* e o *Microsoft Azure Platform*. Por fim, propuseram o *framewok AzureMapReduce* para implementar aplicações de mapeamento e redução em ambientes de nuvem, no qual, comprovam uma eficiência de aplicações HMR implementadas em nuvem, similar àquelas implementadas em *clusters* tradicionais.

Para analisar o comportamento da comunicação em aplicações HMR (NEVES; ROSE; KATRINIS, 2015) desenvolveram o *MRemu* uma ferramenta que permite emular a comunicação reproduzindo-a em diferentes cenários, topologias e configurações, a partir de arquivos de *logs* de execução em servidores reais. Essa ferramenta foi utilizada no presente trabalho para alterar o algoritmo TCP utilizado na comunicação e introduzir a ativação da VCC. Mais detalhes sobre seu funcionamento e implementação são apresentados na Seção 4.2.1.

A aplicação HMR foi utilizada por (DEAN; GHEMAWAT, 2008) em um aglomerado de *clusters* do *Google* para criar uma interface que paraleliza e distribui tarefas de aplicações que demandam computação em larga escala, utilizando computadores convencionais. Os trabalhos até então apresentados que utilizaram a virtualização, o fizeram utilizando MVs, Já em (RISTA DALVAN GRIEBLER, 2017) realizou-se experimentos utilizando recursos de virtualização baseados em contêineres visando melhorar o desempenho da rede ao executar aplicações HMR em nuvens IaaS virtualizadas. Foram utilizadas técnicas para agregação de fluxos de comunicação como forma de aumentar a largura de banda disponível entre os *hosts* que compõem o *cluster*.

3.3.3 Outras Técnicas de Isolamento e Gerenciamento de Fluxo

Nesta Seção, na Tabela 3 é apresentado os trabalhos que foram relevantes e deram sustentação bibliográfica para esta dissertação, mas que, não estão diretamente focados em virtualização, controle de congestionamento, ou aplicações HMR.

Artigo	**Contribuição**
(ZAHAVI et al., 2016)	Propõem o Links como Serviço (LaaS) onde o cliente contrata um *link* dedicado e pode gerenciar seu uso.
(SOUZA et al., 2017)	Abordam os desafios do uso do SDN em nuvens IaaS.
(AL-FARES et al., 2010)	Apresentam a ferramenta *Hedera*, que possibilita um balanceamento de carga de forma dinâmica.
(NOORMOHAMMADPOUR; RAGHAVENDRA, 2017)	Fazem um estudo abordando o impacto da topologia na escalabilidade e balanceamento de carga.

Tabela 3 – Trabalhos relacionados com Técnicas de Isolamento e Gerenciamento de Fluxo.

Para garantir autonomia do cliente no uso da nuvem, (ZAHAVI et al., 2016) propuseram uma nova abstração de serviços de nuvem, denominada LaaS, com isolamento entre os enlaces de comunicação virtualizados. O cliente pode optar por in-

troduzir no enlace o algoritmo de controle de congestionamento que melhor atende as necessidades de sua aplicação, ou seja, a decisão não pertence ao provedor (diferente da abordagem analisada no presente trabalho).

Um dos desafios para a expansão da virtualização se refere a rigidez dos recursos computacionais. O SDN é um conceito que oferece flexibilidade na expansão da rede, uma vez que separa o *plano de controle* do *plano de dados*. Esta característica centraliza o tráfego para o hipervisor monitorar, no entanto, muitos desafios ainda permanecem. (SOUZA et al., 2017) abordam os desafios para a alocação de *Virtual Infrastructure* (VI) em nuvens IaaS utilizando SDN, enquanto (AL-FARES et al., 2010) propuseram a ferramenta *Hedera*, um recurso de agendamento de dinâmico de fluxo que consegue uma eficiência de 96% superior ao balanceamento de carga estático. O plano de controle de dados é centralizado e usa uma topologia *Fat-tree* habilitando o encaminhamento dos fluxos para múltiplos caminhos utilizando o recurso ECMP.

Com relação a topologia de redes em ambiente de DC, (NOORMOHAMMADPOUR; RAGHAVENDRA, 2017) desenvolveram uma pesquisa que aborda as vantagens e desvantagens de algumas topologias de rede no que se refere a escalabilidade e balanceamento de carga, propriedades, técnicas e objetivos de controle de tráfego. Os autores indicam que o problema de falta de equidade entre diferentes fluxos TCP não foi totalmente solucionado. As técnicas apresentadas para solucionar o problema envolvem configurações prévias e complexas. Conforme discutido na Seção 3.1 é importante ressaltar que outras técnicas de tradução podem ser aplicadas a VCC, sendo uma escolha de configuração do provedor. Muitas delas dependem de permissões na MV do inquilino por serem intrusivas como, por exemplo, a escrita na memória da MV do inquilino. Outras técnicas, como a *bufferização*, aumentam a necessidade de processamento no hipervisor ao gerenciar os pacotes em *buffer* e podem provocar um aumento na latência. Aplicações que demandam elevado poder computacional, como as aplicações *Big Data*, sofrem degradação no desempenho quando executadas em plataformas virtualizadas.

3.3.4 Discussão

Dentre os trabalhos apresentados o primeiro grupo, apresentado na Tabela 1 mostram que o recurso de identificar uma provável ocorrência de congestionamento monitorando filas com o uso do gerenciamento da fila RED em conjunto com a marcação efetuada pelo núcleo da rede com o ECN bem como a utilização da VCC já está em uso. Por conseguinte o grupo apresentado na Tabela 2, constatam que a aplicação HMR tem uma participação expressiva no processamento de grandes volumes de dados, como é o caso de aplicações *Big Data* em DCs. Por fim, na Tabela 3 foram apresentados artigos que analisaram a influência da topologia no desempenho da

rede. Os artigos abordaram questões como escalabilidade, balanceamento de cargas e isolamento de fluxos.

Em geral, os autores destes trabalhos analisaram a aplicação em um contexto levando em conta as peculiaridades de processamento e comunicação, sem contemplar as especificidades do algoritmo TCP utilizado. Este trabalho no entanto, procura caracterizar o desempenho da aplicação HMR, levando em conta a condição do algoritmo TCP para esta aplicação em três situações distintas: *(i)* mesmo algoritmo que as demais aplicações compartilhando a rede (atualizado, consciente de ECN); *(ii)* com algoritmo TCP desatualizado (ou seja que não reconhece ECN) e *(iii)* ativando a VCC situação na qual o hipervisor introduzirá informações de notificação explícita de congestionamento, ECN. Nos três casos a aplicação que irá competir com o tráfego HMR está configurado com algoritmo de controle de congestionamento atualizado, ou seja, que reconhece o recurso de notificação de congestionamento ECN.

3.4 CONSIDERAÇÕES PARCIAIS

A virtualização tem se tornado uma realidade para atender as demandas de negócios atualmente. Estações de trabalho, servidores e a rede que interconecta todos esses equipamentos podem ser virtualizados, visando otimizar a ocupação da estrutura física existente.

As tecnologias apresentadas nos trabalhos relacionados evidenciam os recursos que dão suporte ao processo de virtualização do controle de congestionamento. Em paralelo, o ECN age na prevenção de perdas de pacotes marcando aqueles que excedam a um limite estabelecido de ocupação de fila indicando ao emissor que deve reduzir o envio na proporção dos pacotes marcados até que a ameaça de congestionamento seja eliminada. O RED é o recurso que permite que o administrador da rede configure o limite de ocupação e a probabilidade de marcação dos pacotes.

Essas tecnologias já estão disponíveis em equipamentos comutadores como *switches* e os SOs virtualizados. Logo, barreiras tecnológicas para aplicar a virtualização de controle de congestionamento estão sendo gradualmente superadas. Entretanto, as análises inicialmente realizadas sobre VCC são limitadas em escala e representatividade. Diante do exposto, o próximo capítulo apresenta uma caracterização profunda do impacto da virtualização do controle de congestionamento sobre a aplicação HMR.

4 CARACTERIZAÇÃO DO DESEMPENHO DO HMR SOBRE VCC

A principal contribuição do trabalho é descrita no presente capítulo. Inicialmente, a Seção 4.1 apresenta as métricas coletadas para elaborar a caracterização. A carga experimental e a ferramenta responsável pela emulação da comunicação HMR são apresentadas na Seção 4.2. Na sequência, a Seção 4.3 descreve os cenários experimentais utilizados. A discussão individual dos resultados obtidos é apresentada na Seção 4.4, enquanto uma análise combinada é discutida na Seção 4.5. As considerações do capítulo são elencadas na Seção 4.6.

4.1 MÉTRICAS PARA CARACTERIZAÇÃO

Os experimentos objetivam o estudo da aplicação de VCC para virtualizar o tráfego TCP oriundo de MVs não otimizadas que hospedam aplicações HMR. Para analisar a aplicabilidade da VCC, três métricas foram selecionadas e coletadas:

- O **tempo de execução do HMR** é um indicativo direto da visão do cliente sobre o serviço hospedado. Como a reconfiguração da conexão e tradução da comunicação realizadas pela VCC ocorrem de forma transparente, o cliente final, proprietário da aplicação HMR, simplifica a visão sobre a qualidade de serviço através do tempo total necessário para executar a aplicação.
- Em segundo, o **volume de pacotes descartados pelos *switches*** explica parcialmente a variação no tempo de processamento de HMR, decorrente, em grande maioria, ao reenvio de pacotes devido ao esgotamento de *buffers* em *switches*. Ainda, o descarte de pacotes é um indicativo auxiliar da eficácia das regras para marcação de pacotes aplicadas nos *switches*.
- A **quantidade de pacotes na fila** representa a ocupação dos *switches* diante do tráfego aplicado. Uma ocupação elevada contribui para o aumento da latência.

Inicialmente, os resultados obtidos para as métricas elencadas são individualmente analisados (Seção 4.4). Posteriormente, uma análise combinada (Seção 4.5) fomenta a discussão e aponta as principais observações.

4.2 EMULAÇÃO DO TRÁFEGO HMR

O processamento massivo de *Big Data* efetuado com HMR requer um elevado poder computacional para acomodar totalmente o volume de dados e as operações

realizadas. Diversos artigos apresentaram a caracterização do tráfego HMR apontando os requisitos para o cenário analisado (discutidos na Seção 2.3.1). Os resultados apresentam contribuições válidas quando individualmente analisados, entretanto, para uma real reprodução e análise futura, seria necessário acesso ao *hardware* hospedeiro (apresentado na Seção 2.3). Tal fator era limitante para a realização de pesquisas sobre a comunicação de processos HMR em larga escala.

Recentemente, a comunidade desenvolveu um conjunto de emuladores e simuladores de tráfego para HMR (NEVES; ROSE; KATRINIS, 2015; HANDIGOL et al., 2012; WANG et al., 2009). Em geral, as ferramentas oferecem resultados fidedignos aos obtidos em *hardware* real. Dentre os emuladores disponíveis, a presente caracterização é realizada sobre MRemu (NEVES; ROSE; KATRINIS, 2015) usando rastros reais de uma execução prévia de HMR sobre um aglomerado de produção.

4.2.1 Emulador MRemu

O emulador MRemu é utilizado neste trabalho para reproduzir aplicações HMR. Para definir o padrão de comunicação, MRemu utiliza rastros e *logs* reais coletados em DCs de produção. Os rastros guiam a execução da ferramenta, reproduzindo a criação e escalonamento das tarefas de mapeamento de redução.

De posse dos rastros de processamento e comunicação, MRemu emula a comunicação sem efetivamente realizar o processamento, permitindo uma análise da comunicação HMR próxima ao cenário original (NEVES; ROSE; KATRINIS, 2015). Ainda, MRemu permite a alteração da topologia, desde que o número de servidores processadores não seja alterado.

Por ser baseado em SDN, os autores desenvolveram MRemu diretamente sobre a ferramenta *Mininet*, que é um sistema para prototipação de grandes redes, cuja escalabilidade, permite a adição de milhares de *switches* utilizando apenas um laptop. É flexível permitindo que funcionalidades da rede possam ser definidas em *software* usando diferentes linguagens e SOs com a qual o usuário está familiarizado. O gerenciamento e execução da rede ocorre em tempo real podendo interagir com redes externas. Ainda, a pilha de protocolo utilizada em testes pode ser aplicada em redes reais sem nenhuma modificação (LANTZ; HELLER; MCKEOWN, 2010).

Conforme representado na Figura 10 a ferramenta MRemu é constituída basicamente de dois blocos. O primeiro é a simulação do DC no qual a ferramenta Mininet é a base para construir a topologia da rede do DC, utilizando para tal as bibliotecas para construir e monitorar a rede e iniciar a aplicação que pode ser, por sua vez uma reprodução simulada de rastros de execução do Hadoop em DCs reais ou a geração sintética de trabalho HMR. Por sua vez o segundo bloco representa a estrutura do

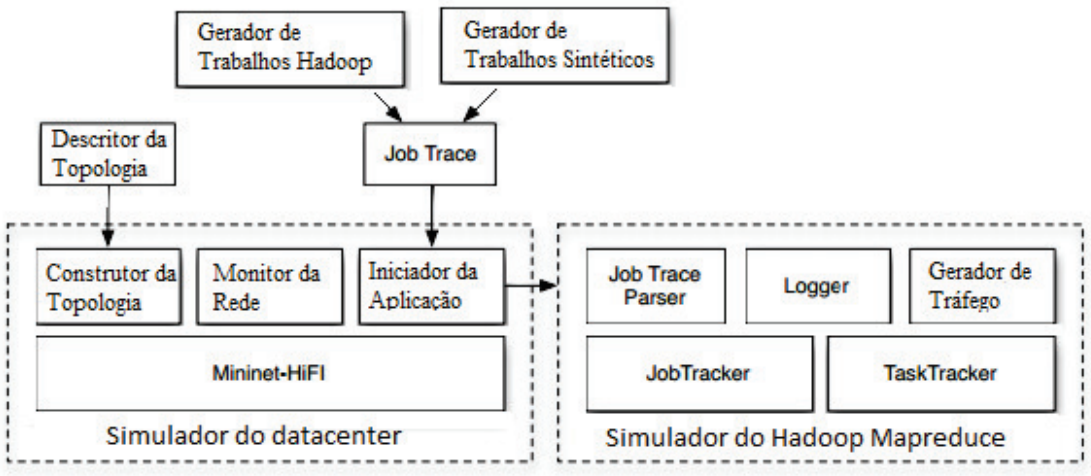


Figura 10 – Diagrama da Arquitetura do MRemu, adaptado de (NEVES; ROSE; KATRINIS, 2015).

simulador, no qual são executados os elementos do HMR como o *Job Tracker* que coordena toda a execução e o *Task Tracker* que distribui as tarefas de mapeamento e redução entre os servidores do cluster, além dos módulos gerador de tráfego e um registro de logs.

4.2.2 Rastro de Execução e Ambiente Hospedeiro

Na Figura 11 é exposto como era a topologia original da aplicação HMR e como foi introduzido o tráfego de background.

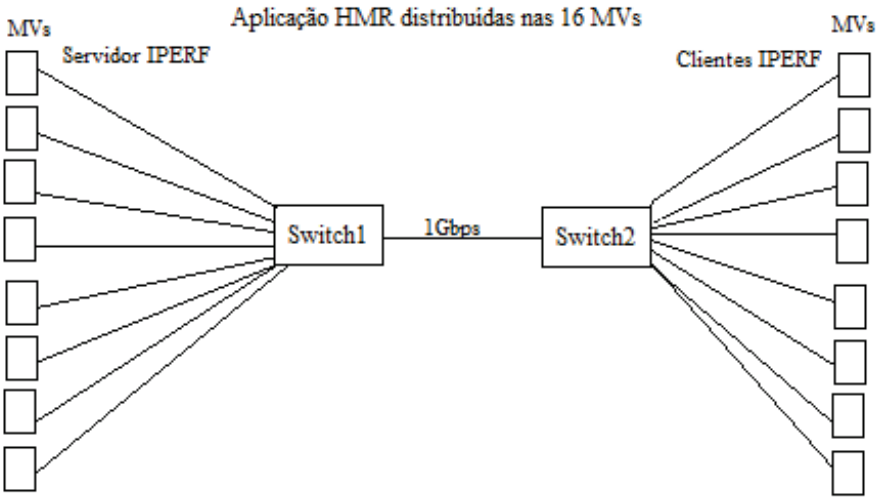


Figura 11 – Topologia do Cenário Origrinal.

O aglomerado que originou os rastros utilizados para a análise da presente seção era composto por 16 servidores, equipados com 12 núcleos $x86\_64$ e 128 GB RAM, interconectados por um enlace de 1 Gbps. O tamanho dos pacotes processados

pelo HMR são de dois tamanhos distintos: 7MB e 9MB. De acordo com a discussão no Capítulo 4, para representação no cenário virtual, com Mininet, 16 hospedeiros virtuais foram utilizados, interconectados por uma topologia *Dumbbell* composta por 2 *switches*. *Dumbbell* é uma excelente abstração dos gargalos que ocorrem em diversos momentos nas topologias revisadas no Capítulo 2. Cada *switch* conecta 8 servidores com enlaces individuais de 1 Gbps. As 16 MVs podem ser mapeados pelo rastreador de trabalho do HMR. O gargalo de comunicação é caracterizado pelo enlace de 1 Gbps que interconecta os *switches*.

Quando necessário, o tráfego de *background* foi introduzido com a ferramenta *iperf*. No *switch1* estão conectados as MVs que foram configuradas como o *servidor* do tráfego *iperf*. Na outra extremidade do enlace no *switch2* são conectadas as máquinas que serão *clientes*.

Quanto ao hospedeiro físico, Mininet foi executado em uma máquina virtual com SO Ubuntu 14.04, 2 vCPUs (dedicadas) e 8 GB RAM, hospedada na nuvem computacional do Laboratório de Processamento Paralelo e Distribuído (LabP2D) da UDESC. Por fim, os recursos de processamento da aplicação HMR não representaram gargalos em nenhum momento, pois MRemu apenas emula a comunicação, sem efetivamente efetuar o processamento.

4.3 CENÁRIOS EXPERIMENTAIS

Para composição dos cenários experimentais, três conjuntos de parâmetros foram alterados para verificar o impacto perante as métricas selecionadas (Seção 4.1). Os conjuntos são individualmente apresentados.

4.3.1 Variantes do TCP

O primeiro conjunto de parâmetros compreendeu a combinação de MVs com TCP não otimizado, TCP otimizado e com VCC, representando as variantes de compartilhamento dos recursos de comunicação (sobretudo do gargalo). Ou seja, para composição do cenário experimental, três configurações TCP foram utilizadas: *(i)* execução com TCP não otimizado sem suporte ao ECN; *(ii)* com TCP otimizado pelo DC, com suporte ECN; e *(iii)* VCC. O tráfego de *background* executou com suporte ao ECN em todos os cenários, em razão de sua finalidade representativa de aplicações atualizadas hospedadas em DC.

4.3.2 Configuração da Marcação de Pacotes

O segundo conjunto de parâmetros diz respeito as configurações RED dos *switches* intermediários para representar as abordagens de configuração comumente

utilizadas por provedores e administradores de DC (resumidos na Tabela 4). As configurações usadas para o RED são oriundas da bibliografia (ALIZADEH et al., 2010; CRONKITE-RATCLIFF et al., 2016), e estão resumidas na Tabela 4.

Configuração	min	max	limit	burst	prob
*RED*1	90000	90001	1M	61	1,0
*RED*2	30000	90000	400K	55	1,0

Tabela 4 – Configurações RED para marcação nas filas dos *switches*.

Por padrão, a literatura indica que a marcação de congestionamento deve ser indicada no instante em que a formação da fila atinge um limite específico (WU et al., 2012). Quando percebe o congestionamento, o emissor reduz a janela na proporção de pacotes ECN marcados. Em suma, o remetente mantém uma estimativa dos pacotes marcados e atualiza seu valor a cada RTT (ALIZADEH et al., 2010). A primeira configuração dos parâmetros na Tabela 4 utilizando o padrão do DCTCP, identificado por *RED1*: o valor do parâmetro *min* está elevado, no entanto não existe uma diferença entre *min* e o max. Todos os pacotes que excederem o limite mínimo serão marcados pois o parâmetro *prob* está configurado para marcar 100% dos pacotes. Na configuração identificada como *RED2* o limite menor, ou seja o *min* tem um valor inferior ao valor do limite máximo representado pelo parâmetro *max*. Nesta configuração existe um intervalo maior de marcação de pacotes que notificarão a ocorrência de congestionamento. A diferença entre as duas configurações também é evidente no tamanho do limite e do parâmetro *burst*.

4.3.3 Tráfego de *Background*

O volume de comunicação entre os pares cliente-servidor, responsáveis pela produção do tráfego TCP de *background*, foi organizado em experimentos conforme representado na Tabela 5. Os experimentos 1 e 2 apresentam uma evolução de carga similar, com a diferença de que cada experimento apresenta um número diferente de pares comunicantes, 1 e 4 respectivamente. O tráfego iniciou com 2MB, aumentando a carga exponencialmente até 32GB (ou seja, foram analisadas transferências com 2MB, 4MB, 8MB, 16MB, 32MB, 64MB, 128MB, 512MB, 1GB, 2GB, 4GB, 8GB, 16GB e 32GB).

No terceiro experimento, não foi limitada a carga de tráfego TCP de *background*, sendo que cada par comunicante enviou o máximo permitido pela aplicação (e gargalo). A diferença neste experimento é o aumento gradativo do número de pares (2, 4 e 8 pares).

A Tabela 5 resume o cenário experimental. Todos os experimentos foram executados alterando as configurações do TCP: otimizado, não otimizado e virtualizado, Seção 4.3.1. A carga experimental dos pares comunicantes foi variada para os experimentos 1 e 2, enquanto o experimento 3 não apresenta limitação. É possível observar

que os cenários experimentais cobrem as sugestões previamente caracterizadas e discutidas pela literatura (Seções 2.3.1 e 3.3). Ainda, os cenários propõem a análise com cargas variadas de tráfego de *background* e configurações distintas para a marcação nos *switches*.

Descrição	**Pares**	**Carga**	**Configuração TCP**
Experimento 1	1 par	2MB a 32GB	otimizado; não otimizado e virtualizado.
Experimento 2	4 pares	2MB a 32GB	otimizado; não otimizado e virtualizado.
Experimento 3	2,4 e 8 pares	Sem limitação	otimizado; não otimizado e virtualizado.

Tabela 5 – Configuração dos experimentos realizados

4.4 RESULTADOS EXPERIMENTAIS E CARACTERIZAÇÃO

Os resultados apresentados foram coletados em 10 execuções para cada combinação de parâmetros. Nos gráficos que mostram a mediana dos valores, o desvio padrão com um intervalo de confiança de 95% é apresentado. É importante frisar que devido a grande variabilidade apresentada nos resultados de um cenário em relação ao outro, a escala dos gráficos não foi uniformizada, entretanto, não constitui um fator limitante para a caracterização e análise dos resultados.

Os gráficos que representam o tempo de execução da aplicação HMR e a perda de pacotes são apresentados como diagramas de caixa (*boxplot*), conforme exemplificado pela Figura 12. Este tipo de gráfico é constituído de uma haste que representa na extremidade inferior o valor mínimo enquanto que a extremidade superior representa o valor máximo. O círculo fora da área da haste representa valores atípicos ou discrepantes denominado *outlier*, reforçando que pode aparecer acima do limite superior da haste bem como abaixo do limite inferior. O retângulo que sobrepõe a haste traz consigo três valores: O primeiro *quartil* que é o seu contorno inferior; segundo *quartil* ou *mediana* e o traço central, e por último o terceiro traço que delimita o limite superior ou terceiro *quartil*. A análise deste trabalho foi baseada na comparação do segundo *quartil* ou mediana dos gráficos que representam os diferentes experimentos e cenários executados.

Nos gráficos apresentados neste trabalho eixo X representa a carga de *background* competindo com a aplicação HMR. Por sua vez, o eixo Y representa o tempo em segundos para finalizar o processamento completo com cada carga com a qual foi executado.

Já nos gráficos que representam a perda de pacotes são similares aos gráficos descritos anteriormente, com a diferença de que o eixo Y ao invés de representar o

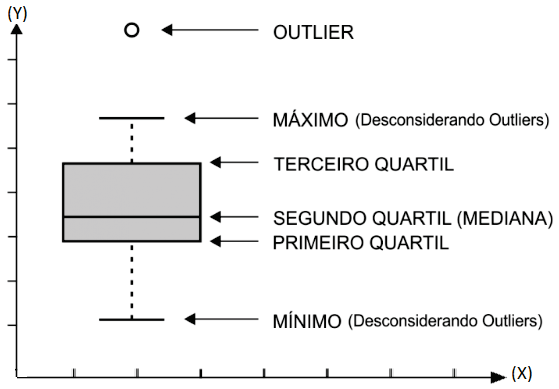


Figura 12 – Interpretação de um diagrama de caixa, adaptado de (BOXPLOT, 2018).

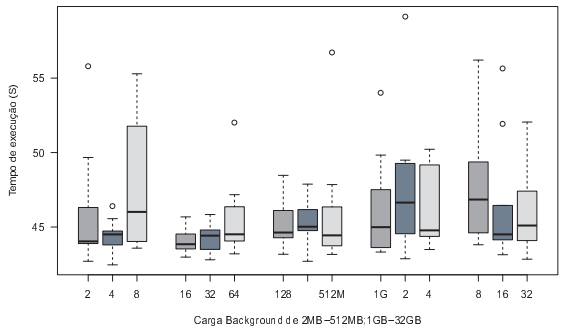
tempo em segundos, representa a perda de pacotes em *bytes* para cada uma das cargas introduzidas no experimento. Ainda, as filas são representas por um gráfico de função de distribuição acumulada (*Cumulative Distributed Function* (CDF)), no qual o eixo Y representa a probabilidade de ocorrência de filas e o eixo X representa a quantidade de *bytes* ocupando a fila.

4.4.1 Análise do Experimento 1

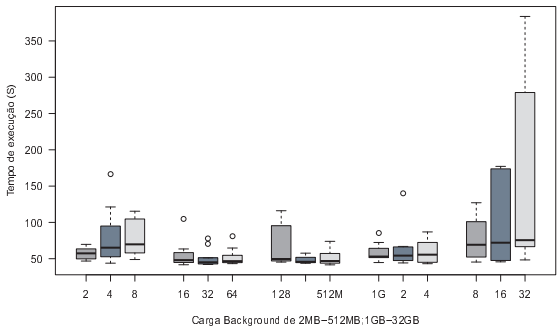
O experimento 1 compreende a execução de um par cliente-servidor gerando tráfego de *background* variando a carga conforme representado na Tabela 5. O HMR foi executado alterando as configurações do TCP: *(i)* cenário atualizado (com ECN); *(ii)* cenário legado (sem ECN), e *(iii)* cenário virtualizado (com VCC ativada). O tempo de execução, a perda de pacotes e a formação de filas são individualmente analisados em sequência.

4.4.1.1 Tempo de Execução

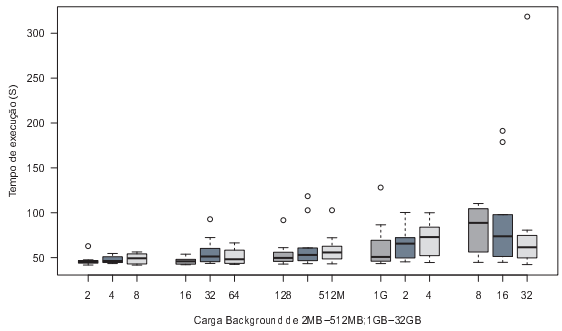
O tempo de execução da aplicação HMR quando o gargalo de comunicação com apenas um par cliente-servidor de tráfego de *background* é apresentado na Figura 13. Inicialmente, a Figura 13(a) apresenta os resultados utilizando a configuração de gerenciamento de fila RED1. Constata-se que, mesmo variando a carga de competição, a maioria das execuções apresentou um tempo de execução entre 40 e 50 segundos, com exceção da execução com carga de 8MB. Ainda, é importante ressaltar a presença de valores atípicos (*outliers*) demostrando que mesmo em cenários com algoritmos TCP homogêneos (e atualizados, com suporte ao ECN) a variabilidade está presente. Em suma, a variação no intervalo é justificada pela agressividade na retomada do aumento da janela de transmissão que pode comprometer aplicações mais sensíveis ao congestionamento, aumentando por fim o tempo de execução.



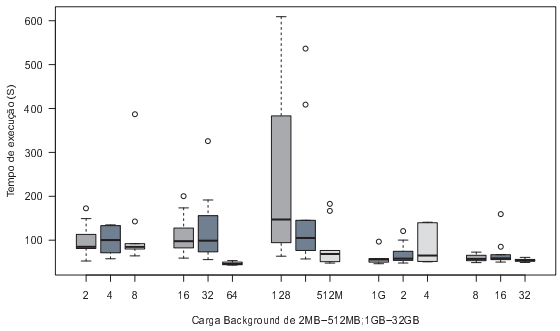
(a) Com ECN - RED1.



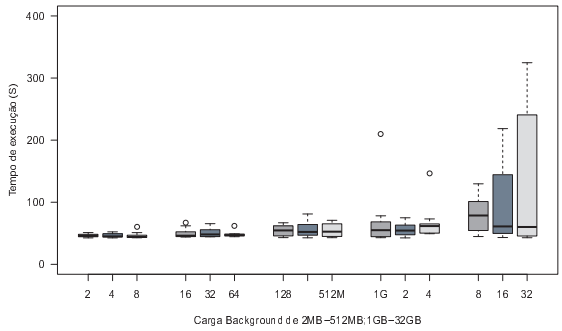
(b) Sem ECN - RED1.



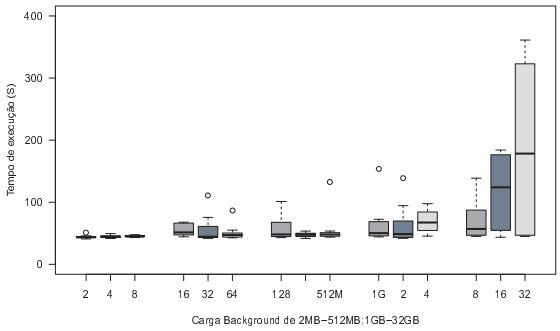
(c) Virtualizado - RED1.



(d) Com ECN - RED2.



(e) Sem ECN - RED2.



(f) Virtualizado - RED2.

Figura 13 – Tempo de execução do HMR compartilhando o gargalo com um par cliente-servidor de tráfego de *background*.

No cenário sem ECN (conforme a Figura 13(b)) o limite superior do intervalo de variação do tempo de execução aumentou exponencialmente para cargas competidoras com tamanhos de 8GB, 16GB e 32GB. Isso demonstra que o maior volume que ocupou o gargalo pertence aos fluxos de *background* que utilizam ECN para notificar congestionamento, prevenindo a perda de pacotes. No entanto, a aplicação HMR neste contexto não utiliza ECN para notificar congestionamento, logo a perda de pa-

cotes é mais frequente (posteriormente analisado) e, portanto realiza retransmissões de pacotes perdidos impactando o tempo de execução.

Ao aplicar a técnica de VCC para a configuração RED1, a Figura 13(c) demonstra que a virtualização conseguiu melhorar (diminuir a variabilidade) o tempo de execução da aplicação que está sendo traduzida, com todas as cargas de fluxo nativo consciente de ECN, inclusive para grandes volumes como os testes com 16GB e 32GB. Muito embora pode ser notada a presença de vários *outliers*.

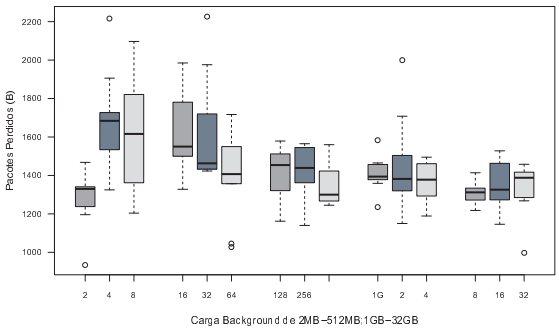
Com RED2 no cenário atualizado (com suporte ao ECN), Figura 13(d), o tempo de execução oscilou entre de 50 e 60 segundos. Entretanto, foi possível estabelecer uma relação entre o tamanho da carga e a variabilidade do tempo de execução, uma vez que ela é mais expressiva com cargas menores, como na execução com 32MB, e também com cargas maiores como nas execuções com as cargas de 1GB, 2GB e 32GB. Possivelmente a marcação de pacotes agressiva realizada por RED2 normalizou o tempo de execução. Para corroborar a análise, a execução sem suporte ao ECN (Figura 13(e)) indicou que para cargas acima de 1GB ocorre uma maior variabilidade no tempo de execução, demonstrando que a carga de dados afeta o desempenho do tempo de execução da aplicação HMR em cenários com controle de congestionamento heterogêneos.

Por fim, no cenário virtualizado com RED2, Figura 13(f), o tempo de execução apresentou uma variabilidade superior ao cenário legado, mais expressivamente em execuções com maiores cargas. Isso demonstra que a aplicação HMR em cenários com VCC e RED2 tem o seu desempenho comprometido quando a carga concorrente é mais expressiva.

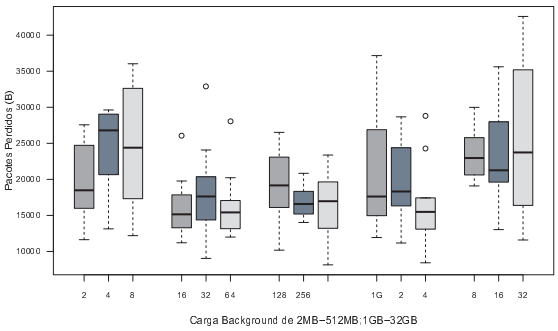
4.4.1.2 Perda de Pacotes

Analisando as perdas de pacotes em um cenário com 1 par comunicando tráfego de *background* (configuração com RED1) pode-se verificar que as perdas no cenário com suporte ao ECN, representado na Figura 14(a), apresentam uma variabilidade entre as leituras com algumas execuções apresentando *outliers* (menores que 500 pacotes e acima de 2200 pacotes). A coleta mostrou uma predominância da ocorrências de perdas de pacotes entre 1300 e 1800. A execução com 2MB mostrou a menor perda, enquanto a execução com 8MB apresentou uma maior perda, seguida da coleta com 16MB, 32MB e 4MB. As demais execuções demonstraram um resultado similar. A caracterização indica que com maior carga competidora, os resultados mostraram uma menor perda, possivelmente por fluxos maiores conseguirem estabilizar o tamanho da janela de transmissão de dados.

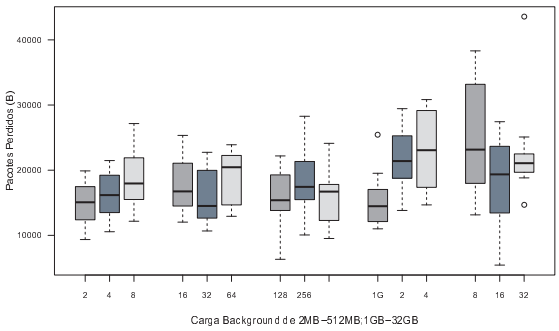
No cenário sem suporte ao ECN para HMR (ainda com RED1), os resultados



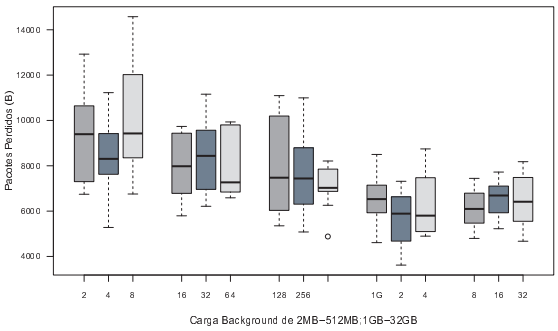
(a) Com ECN - RED1.



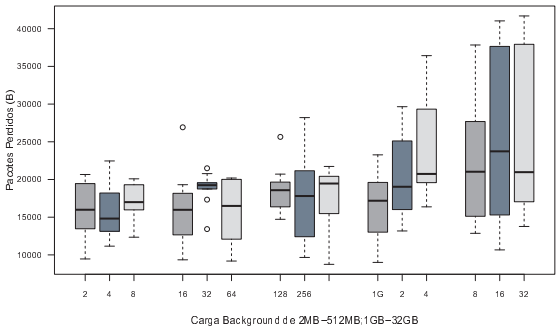
(b) Sem ECN - RED1.



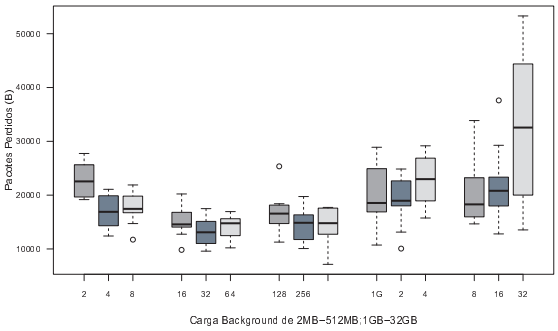
(c) Virtualizado - RED1.



(d) Com ECN - RED2.



(e) Sem ECN - RED2.



(f) Virtualizado - RED2.

Figura 14 – Perdas de pacotes com o HMR compartilhando o gargalo com um par cliente-servidor de tráfego de *background*.

indicaram uma perda de pacotes superior aquela observada no cenário anterior (com ECN). Conforme demonstrado na Figura 14(b), a perda de pacotes atingiu um limite superior a 30000 pacotes perdidos e apresentou uma variabilidade maior que a observada no cenário atualizado (com ECN). Esta variabilidade pode ser explicada pelo fato de ter fluxos que antecipam a percepção de congestionamento, no caso os fluxos de *background*, enquanto que os fluxos do HMR dependem da ação das extremidades

comunicantes do TCP para perceber o congestionamento, ou seja, *timeouts* ou ACKs duplicados.

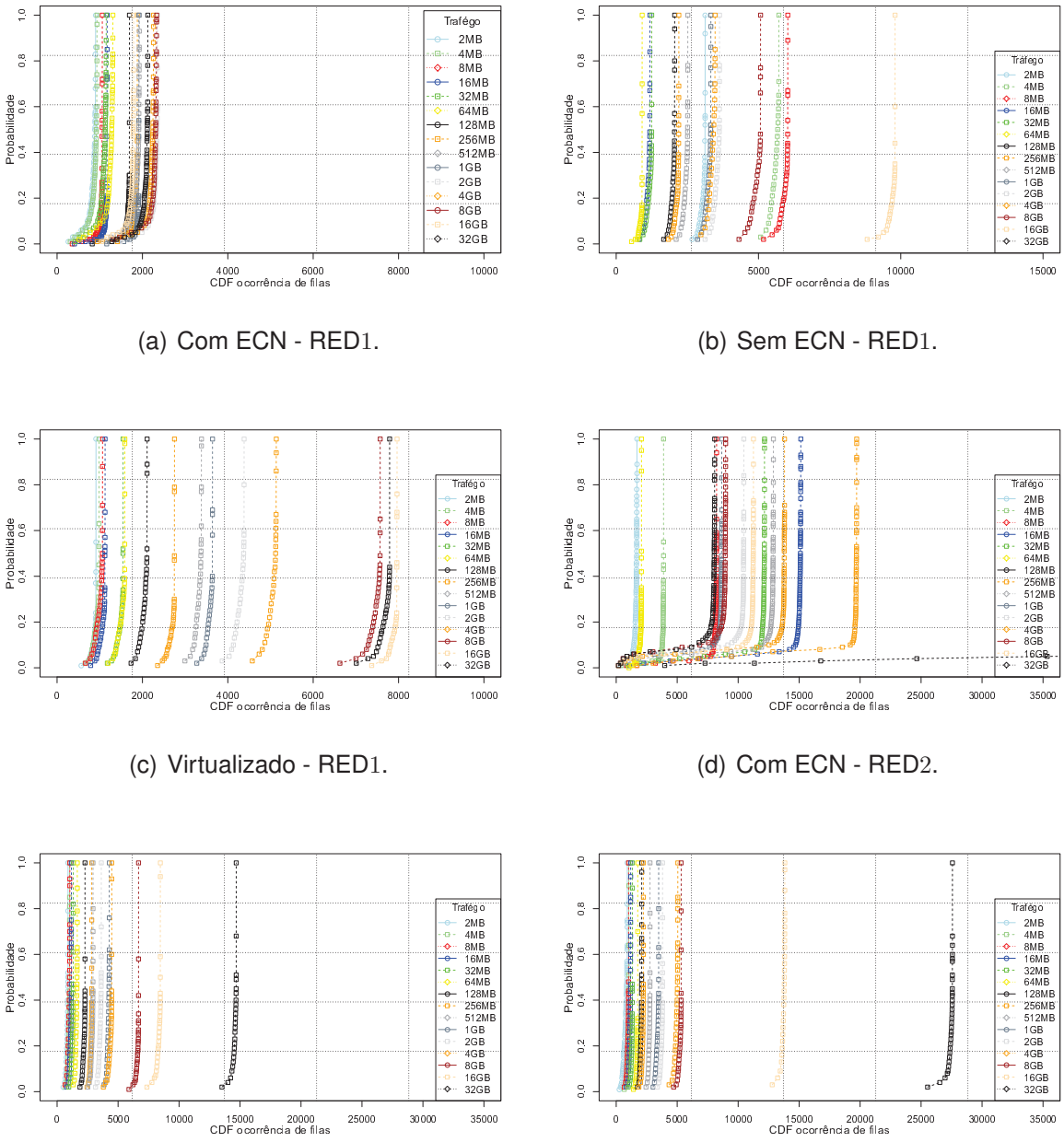
Ao ativar a VCC combinada com a configuração RED1, Figura 14(c), é possível verificar a existência de *outliers* com uma leitura de perda de pacotes acima de 40000. Entretanto, com cargas concorrente entre 2MB e 1GB, a VCC conseguiu diminuir o volume de perda de pacotes, mantendo a concentração de perda entre 12000 e 20000 pacotes. O mesmo é observado com cargas concorrentes de 16GB e 32GB. Surpreendentemente com 4GB e 8GB a virtualização não conseguiu diminuir a perda de pacotes em relação ao cenário legado, ao contrário, aumentou os valores. Possíveis problemas na retomada do tamanho da janela de dados, como a entrada em fase de *slow start* de alguma das conexões TCP com algoritmo New Reno, pode ser a causa desse aumento de perdas (uma correlação com a janela de congestionamento é indicada como trabalho futuro, já que a presente caracterização apontou o potencial problema).

Na sequência, os cenários executados com a configuração de gerenciamento de fila RED2 são analisados. Na Figura 14(d) é exposta a perda de pacotes no cenário com ECN ativado para HMR e para o tráfego de *background*. Com exceção da execução com 2MB de carga concorrente, que apresenta um volume abaixo de 2000 pacotes perdidos, as demais cargas tiveram uma variação de perdas oscilando entre 6000 e 8000 pacotes. Essas cargas corroboram que o TCP New Reno tende a ser agressivo na abertura da janela de dados, inundando o *buffer* do *switch*. Já no cenário legado, Figura 14(e) , observa-se que houve um aumento expressivo do número de pacotes perdidos com uma mediana acima de 15000, praticamente o dobro das perdas constatadas no cenário anterior. Além disso é possível constatar que com cargas maiores a variabilidade em torno da mediana é maior. Ao aplicar a VCC, Figura 14(f), pode-se verificar que não houve uma redução significativa da ocorrência de perdas em relação ao cenário legado, porém manteve uma menor variabilidade.

4.4.1.3 Formação de Filas

A Figura 15(a) representa a probabilidade de formação de fila no *buffer* do *switch* em um cenário atualizado com a configuração RED1 em funcionamento. Com carga de até 64MB é possível manter uma ocupação do *buffer* em torno de 1000 *bytes*. Nas demais execuções com cargas concorrentes, a ocupação da fila é aproximadamente entre 1500 e 2100 *bytes*. Essa uniformidade ocorre nesse cenário porque as duas aplicações que compartilham o gargalo da rede tem o mesmo recurso de controle de congestionamento, ou seja, enviam sinalização ECN de provável ocorrência de congestionamento para as extremidades comunicantes.

Já em ambiente heterogêneo, conforme a Figura 15(b), a formação da fila é



(a) Com ECN - RED1.

(b) Sem ECN - RED1.

(c) Virtualizado - RED1.

(d) Com ECN - RED2.

(e) Sem ECN - RED2.

(f) Virtualizado - RED2.

Figura 15 – Formação de filas com o HMR compartilhando o gargalo com um par cliente-servidor de tráfego de *background*.

mais expressiva e variada, oscilando de 200 *bytes* para mais de 6000 *bytes* de ocupação do *buffer*. Como as aplicações legadas não conseguem alertar as extremidades da ocorrência de pacotes, ocorrem *timeouts* e consequente a diminuição do tamanho da janela e retomada lenta do envio de dados (justificando o tempo de execução previamente analisado).

Na Figura 15(c) pode-se verificar que a granularidade da ocupação da fila com VCC é mais explícita que no cenário legado e atualizado. Ainda, possui variação entre aproximadamente 600 e 8000 *bytes*. Isso demonstra que a VCC sofre um impacto no gerenciamento de fila quando o tráfego virtualizado está competindo com tráfego nativo ECN com maiores cargas.

Na configuração com RED2 para cenário somente ECN, Figura 15(d), o tamanho da fila no *switch* não ultrapassou os 8000 *bytes*. Por sua vez, no cenário legado, Figura 15(e), com cargas concorrentes abaixo de 512MB a ocupação da fila ficou semelhante ao cenário atualizado, porém com cargas maiores teve um aumento expressivo, com destaque para a execução com 32GB cuja ocupação da fila atingiu a marca de 15000 *bytes*. Ao ativar a VCC é possível verificar que com cargas de 16GB e 32GB houve um aumento significativo da ocupação da fila, ou seja, a virtualização potencializou o resultado do pior cenário, no entanto com cargas menores, houve uma leve diminuição da ocupação do *buffer* em relação ao mesmo cenário legado.

4.4.1.4 Principais Observações

Métrica	**Observação**
Tempo de execução	- Com fluxos menores, a configuração RED1 apresenta maior variabilidade no cenário com ECN. - Com RED1, a VCC conseguiu melhorar o tempo em relação ao cenário sem ECN. - Para o TCP sem ECN com cargas concorrentes acima de 8GB, as duas configurações, RED1 e RED2, apresentaram grande variabilidade. - Com VCC ativada e configuração RED2, foi observado um maior tempo de execução do HMR com cargas grandes, acima de 8GB.
Pacotes perdidos	- Maior predominância na configuração RED2 em cenário atualizado TCP. - Em cenário sem ECN e com VCC, RED1 e RED2 apresentam grande variabilidade.
Formação de filas	- Maior predominância na configuração RED2 em cenário atualizado. - Em cenário legado e virtualizado foi maior a ocorrência de filas em RED2 com maiores cargas.

Tabela 6 – Principais observações no experimento 1.

As principais observações do experimento 1 são resumidas na Tabela 6. É importante ressaltar que a carga concorrente influencia de diferentes maneiras na formação da fila, assim como nas perdas de pacotes, conforme analisado na Seção 4.4.1.2, e no tempo de execução, descrito na Seção 4.4.1.1.

4.4.2 Análise do Experimento 2

O experimento 2 compreende a execução de quatro pares cliente-servidor gerando tráfego de *background* e variando a carga conforme representado na Tabela 5.

Este tráfego corresponde a aplicação atualizada, ou seja o algoritmo TCP nesta aplicação tem consciência da notificação de congestionamento pelo núcleo da rede (ECN). O HMR é a aplicação a qual queremos caracterizar o impacto sofrido em ambientes virtualizados, Ele foi executado alterando as configurações do TCP para cenário atualizado (com ECN); cenário legado (sem ECN), e cenário virtualizado (com VCC ativada). Os valores coletados para as métricas são sequencialmente analisados.

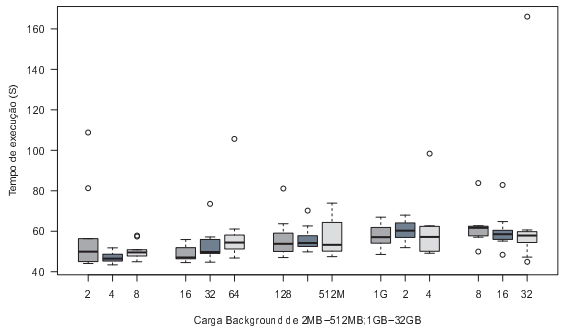
4.4.2.1 Tempo de Execução

Ao realizar análise do tempo de execução quando configurado com RED1 para o cenário atualizado (representado pela Figura 16(a)) constata-se uma variação da mediana entre 45 e 55 segundos. Ainda, alguns *outliers* são observados. Especificamente sobre a caracterização, não é possível verificar um impacto direto da carga de competição perante ao tempo de execução do HMR uma vez que a execução com 2MB apresenta uma mediana e variação similares aos resultados com 512MB ou 4GB, por exemplo. Como a aplicação HMR, neste cenário utiliza o mecanismo ECN para a percepção de congestionamento, tal qual a aplicação de background. Assim o controle de congestionamento é realizada da mesma maneira. Dessa forma, o volume de dados não impactou no tempo de execução.

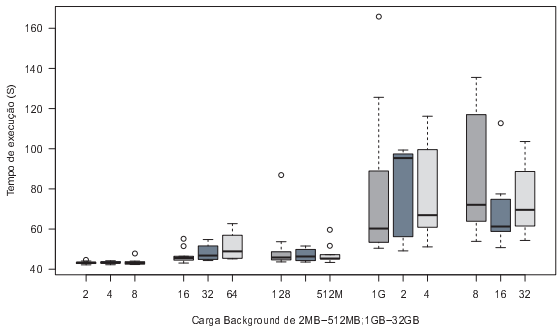
Na Figura 16(b) são apresentados os resultados para a configuração RED1 e TCP sem ECN. Não houve um aumento expressivo do tempo de execução do HMR em relação à carga de fluxos de *background* de 2MB até 512MB. No entanto, com cargas de 1GB até 32GB houve um aumento expressivo no tempo de execução do HMR, chegando a um tempo aproximado de 120 segundos quando competindo com fluxos que trafegam 8GB, o que é praticamente o dobro do tempo de execução percebido no cenário atualizado. Como os tamanhos de pacotes do HMR não ultrapassam 10MB (discutido na Seção 2.3.1), ocorre uma maior latência de enfileiramento quando concorrem com fluxos com volumes maiores e sobretudo com recurso de sinalização de congestionamento ativado.

Avaliando o cenário virtualizado com a configuração RED1, na Figura 16(c), é possível verificar o efeito que a carga de aplicações nativas ECN tem sobre o tempo de execução de aplicações virtualizadas com VCC. Com cargas competidores de 1MB até 256MB, traduzir o algoritmo legado (sem ECN) para o algoritmo consciente de tráfego ECN conseguiu uma melhoria no tempo de execução, porém, em cargas maiores (como nas execuções de 512MB a 32GB) ocorre uma grande variabilidade nos resultados piorando a perspectiva do cliente em relação ao cenário legado. O motivo é diretamente relacionado com a ocupação das filas, como será posteriormente caracterizado.

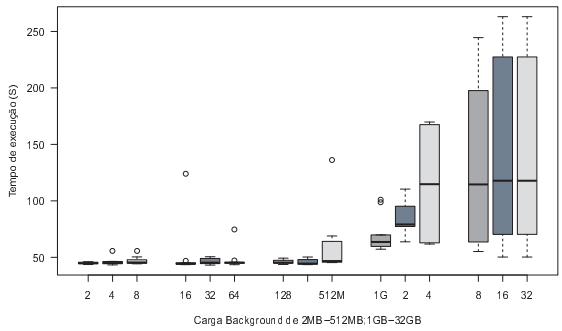
Na sequência, são apresentados os resultados para a configuração RED2. Na



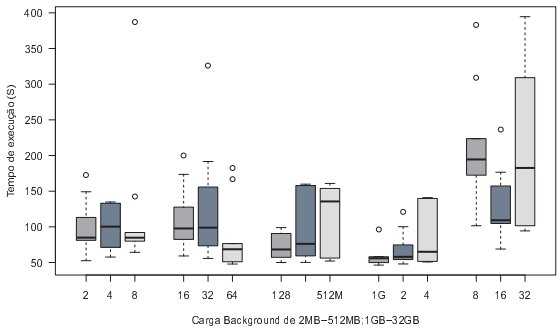
(a) Com ECN - RED1.



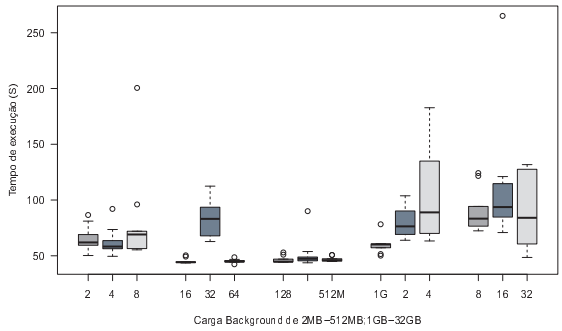
(b) Sem ECN - RED1.



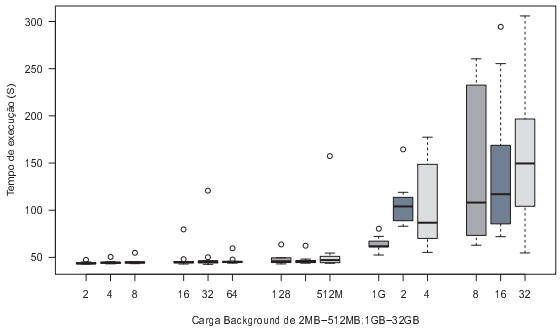
(c) Virtualizado - RED1.



(d) Com ECN - RED2.



(e) Sem ECN - RED2.



(f) Virtualizado - RED2.

Figura 16 – Tempo de execução do HMR compartilhando o gargalo com quatro pares cliente-servidor de tráfego de *background*.

Figura 16(f), com a VCC ativada, o tempo de execução foi menor quando concorrente com fluxos menores que 1GB, no entanto, para cargas maiores o resultado se mostrou pior que o apresentado no cenário legado, Figura 16(e). Novamente fica claro que o tempo de execução do HMR é influenciado pela carga de dados compartilhando a estrutura da rede. Surpreendentemente, na Figura 16(d), que representa o tempo de execução em um cenário atualizado, o tempo de execução da aplicação HMR foi

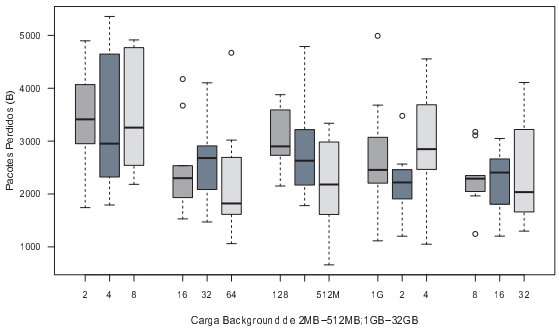
maior que nos cenários legado e com VCC ativada. A configuração RED2 tem como característica um limite mínimo de inicio de notificação de congestionamento equivalente a $1/3$ do limite do mesmo parâmetro para a configuração RED1. Logo, começa a marcar pacotes com uma menor ocupação de fila. Da mesma forma o intervalo de marcação é mais amplo se estendendo por $3x$ o valor do limite inicial. Essa maior granularidade na marcação de pacotes explica a oscilação apresentada neste cenário. Como as duas aplicações disputando a ocupação do gargalo da rede utilizam TCP atualizados, consciente de ECN, eles ajustam suas janelas de acordo com a informação recebida do núcleo da rede. Ou seja as duas aplicações, tráfego de *background* e a aplicação HMR frearam o envio de dados reduzindo o tamanho de suas janelas de dados baseados na predição de congestionamento proporcionada pelo ECN, ou seja, antes da real ocorrência de perdas de pacotes (discutido na próxima seção).

4.4.2.2 Perda de Pacotes

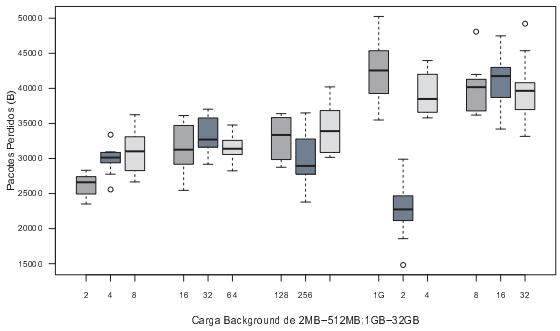
A análise da perda de pacotes neste experimento inicia pelo cenário de marcação com a configuração RED1. A perda de pacotes apresenta uma diferença significativa entre os cenários atualizado, Figura 17(a), legado Figura 17(b) e virtualizado Figura 17(c). Em razão de tal variabilidade não foi possível uniformizar as escalas dos gráficos.

No cenário atualizado, Figura 17(a), a perda se concentra entre 1900 e 4000 pacotes perdidos, apresentando *outliers* em alguns casos. Interessante notar que com cargas competidoras menores (2MB, 4MB e 8MB), o volume perdido foi superior, chegando próximo de 5000 pacotes perdidos. Como todos os fluxos utilizam a informação de congestionamento repassada pelo núcleo da rede, a perda em cargas maiores é menor influenciada pela estabilização da janela de dados. Cargas menores estão mais suscetíveis a agressividade na abertura da janela.

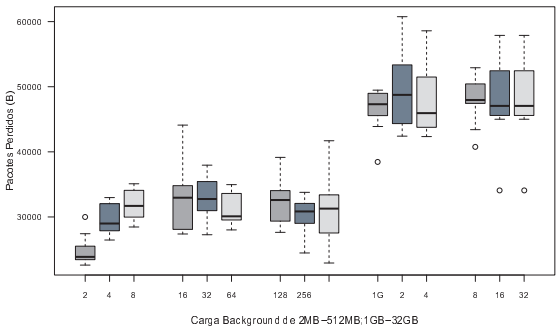
Analisando a Figura 17(b), constatamos que a perda de pacotes aumentou significativamente em relação ao cenário atualizado. Com cargas de 2MB até 512MB a perda se concentrou entre 25000 e 35000 pacotes perdidos. Com cargas maiores de 1GB, 4GB, 8GB, 16GB e 32GB, os valores se mantiveram acima de 35000 pacotes perdidos. Um fato interessante é a execução com 2GB que registrou uma perda abaixo de 25000 pacotes. Como causa desta variabilidade é indicado que conexão tenha conseguido uma marcação mais uniforme do bit ECN, chegando ao *switch* em uma sequência após os pacotes da aplicação legada. O *buffer* foi inicialmente preenchido por pacotes de aplicações legadas que conseguiram neste instante maior vazão por estarem em uma linha abaixo da marcação e descarte do gerenciador de fila RED. Como esse cenário é heterogêneo, a perda de pacotes tende a ser maior já que aplicações com controle ECN tendem a frear o envio de dados baseado na



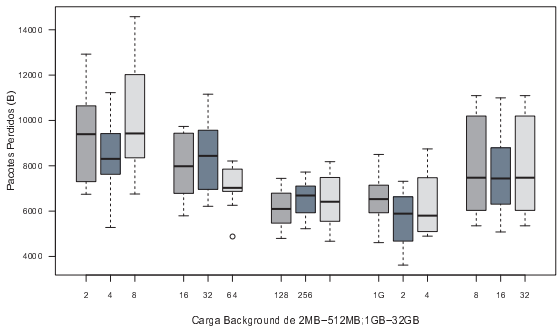
(a) Com ECN - RED1.



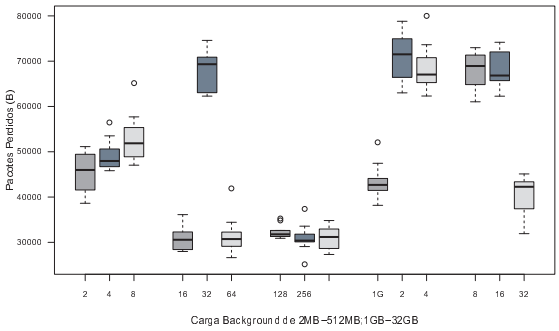
(b) Sem ECN - RED1.



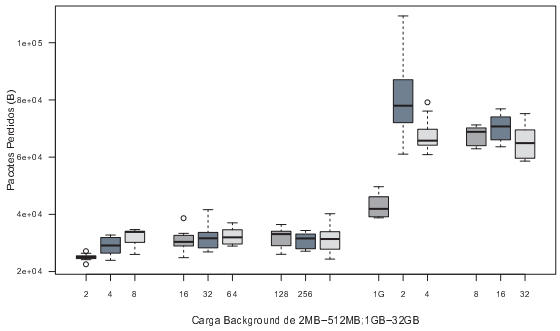
(c) Virtualizado - RED1.



(d) Com ECN - RED2.



(e) Sem ECN - RED2.



(f) Virtualizado - RED2.

Figura 17 – Perda de pacotes com o HMR compartilhando o gargalo com quatro pares cliente-servidor de tráfego de *background*.

marcação, antecipando e evitando a perda de pacotes. No entanto, aplicações que não usam esse recurso tem uma elevada perda de pacotes quando se depara com enlaces congestionados.

Ao ativar a VCC, Figura 17(c), observa-se que ocorre uma diminuição da perda de pacotes quando a carga de fluxos concorrentes nativos ECN está entre 2MB e

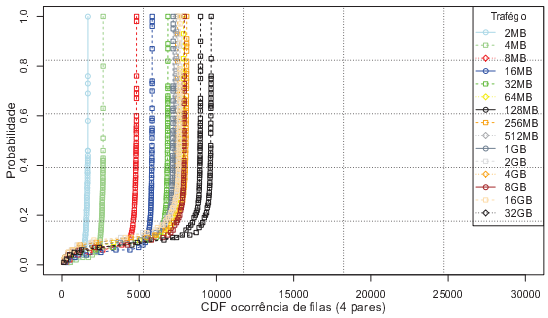
512MB. Para cargas maiores, como observado nas execuções de 1GB até 32GB, a conversão de algoritmo legado para atualizado aumentou o número de pacotes perdidos em relação ao cenário heterogêneo. O benefício da virtualização no controle de congestionamento é influenciado pela carga do fluxo concorrente que não necessita de tradução, ou seja que reconhece ECN nativamente.

Nas Figuras 17(d), 17(e) e 17(f) são apresentadas as perdas de pacotes respectivamente para os cenários atualizado, legado e virtualizado com a configuração de gerenciamento de fila RED2. No cenário atualizado, a perda de pacotes oscila em função da carga concorrente aplicada, mas manteve um teto de ocupação da fila abaixo de 10000 *bytes* (posteriormente caracterizado). Ou seja, o ECN consegue evitar perdas de pacotes notificando antecipadamente a origem a respeito do nível de tráfego dos fluxos na rede. No cenário legado, a oscilação da perda de pacotes varia de 30000 *bytes* até 70000 *bytes*. Entretanto, não é possível relacionar o aumento de perdas com a carga concorrente, uma vez que execuções com carga de 32MB, por exemplo, apresentaram perdas similares às cargas concorrentes acima de 2GB. Neste cenário a perda de pacotes é potencializada para as aplicações que não reconhecem as notificações de formação de filas. Ao ativar a VCC, Figura 17(f), a perda de pacotes apresentou uma diminuição significativa com cargas menores.

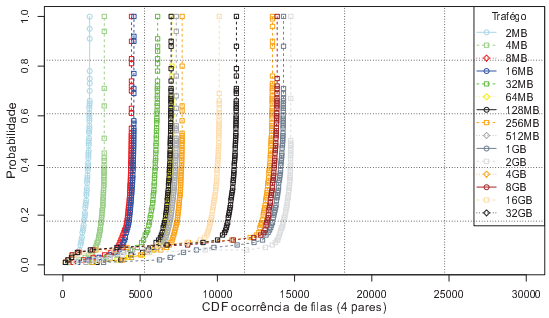
4.4.2.3 Formação de Filas

As ocupações das filas neste experimento são destacadas nas Figuras 18(a), 18(b) e 18(c) representando respectivamente os cenários atualizado, legado e virtualizado com a configuração de marcação de pacotes RED1. O cenário atualizado mostra uma distribuição gradativa das filas em relação ao tamanho da carga injetada pelo fluxo de *background*. Neste cenário, os fluxos da aplicação HMR tem o mesmo algoritmo da aplicação concorrente, portanto a evolução dos fluxos é semelhante. A probabilidade de ocupação da fila não ultrapassou a quantidade de 10000 *bytes*.

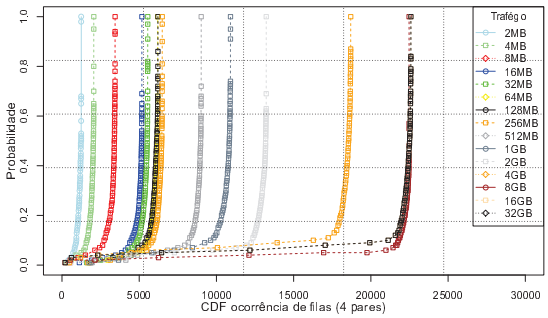
No resultado apresentado na Figura 18(b), a probabilidade de ocupação da fila aumentou em 50% em relação ao cenário atualizado, atingindo aproximadamente 15000 *bytes*. Diferentemente do cenário anterior, não houve uma ocupação gradativa em relação a carga, por exemplo, na execução com 32GB a ocupação ficou em torno de 10000 *bytes* enquanto que com 512MB apresentou uma formação de fila em torno de 15000 *bytes*, ou seja, uma carga menor manteve uma maior ocupação do *buffer*. Neste cenário, o fluxo consciente de ECN consegue abrir sua janela de dados rapidamente em relação ao fluxo que utiliza recursos tradicionais de gerenciamento de congestionamento. Essa dinâmica se reflete na variabilidade da ocupação das filas em relação as cargas de fluxos nativos que controlam o congestionamento com auxilio do núcleo da rede.



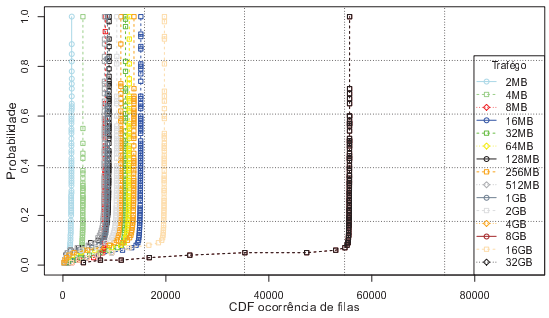
(a) Com ECN - RED1.



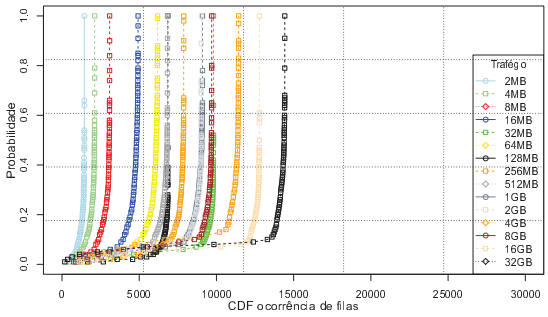
(b) Sem ECN - RED1.



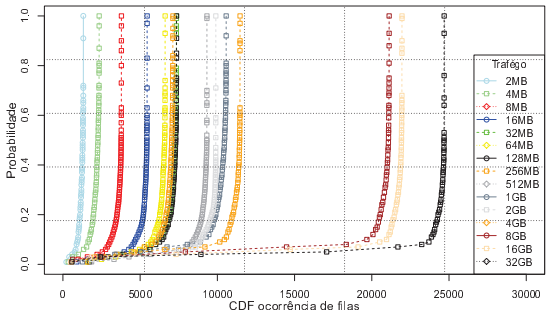
(c) Virtualizado - RED1.



(d) Com ECN - RED2.



(e) Sem ECN - RED2.



(f) Virtualizado - RED2.

Figura 18 – Formação de fila com o HMR compartilhando o gargalo com quatro pares cliente-servidor de tráfego de *background*.

No cenário virtualizado, Figura 18(c), pode-se constatar que com cargas concorrentes de até 256MB, a VCC conseguiu manter a ocupação da fila em um patamar pouco acima do 5000 *bytes*, mostrando em alguns casos um resultado melhor que o cenário atualizado com 8GB e 16GB, por exemplo. No entanto, a medida que cargas concorrentes maiores foram introduzidas, a virtualização mostrou uma variabilidade da ocupação de fila elevando a ocupação para um valor próximo de 25000 *bytes*. Isso

corrobora o que foi concluído anteriormente: que a VCC é suscetível ao tamanho do fluxo de dados e consequentemente impacto no tempo de execução do HMR.

A análise das filas com a configuração RED2 demonstra que HMR executando com TCP otimizado teve uma ocupação da fila granular (Figura 18(d)) com grande variabilidade em relação a carga. Por exemplo, com 32GB a ocupação da fila atingiu aproximadamente 55000 *bytes*. No entanto, com cargas menores manteve um valor inferior a 20000 *bytes*.

No cenário com VCC ativada, Figura 18(f), ocorre uma ocupação de filas similar ao cenário legado, Figura 18(e). Para cargas concorrentes de até 8GB alcança em torno de 10000 *bytes*, porém com cargas maiores a virtualização é marcada por um incremento na formação de filas.

4.4.2.4 Principais Observações

A Tabela 7 resume as principais observações para o experimento 2. Em resumo, é possível observar que conforme aumenta a carga concorrente, o processo de tradução não consegue melhorar o desempenho da aplicação HMR executando sobre TCP legado.

Métrica	**Observação**
Tempo de execução	- Aumento do tempo em cargas maiores com RED1 sem ECN; - Alta variabilidade com RED2 sem ECN; - A virtualização é afetada quando compartilha o gargalo com cargas concorrentes maiores, a partir de 1GB.
Pacotes perdidos	- As perdas são maiores em cenários heterogêneos (HMR sem ECN); - A configuração RED2 apresenta uma grande variabilidade; - A virtualização consegue diminuir perdas com cargas menores, porém aumenta com cargas maiores.
Formação de filas	- A ocupação da fila é maior em cenários heterogêneos (HMR sem ECN); - A virtualização consegue diminuir esta ocupação com cargas menores, porém aumenta com cargas maiores.

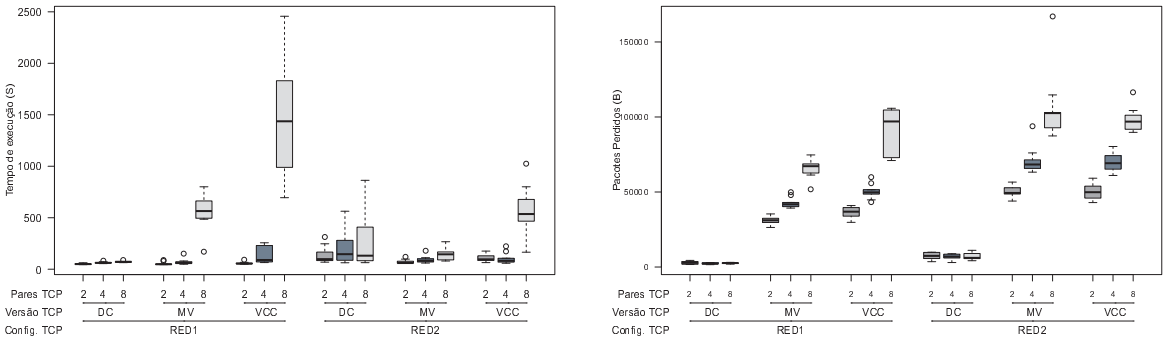
Tabela 7 – Principais observações no experimento 2

4.4.3 Análise do Experimento 3

Neste experimento, ao contrário dos cenários discutidos e caracterizados anteriormente, não houve uma alteração da carga dos pares concorrentes. A variação foi obtida introduzindo mais pares de aplicações atualizadas (que utilizam ECN para notificar congestionamento), competindo pelos recursos de rede. O primeiro turno de teste foi realizado com 2 pares competindo com a aplicação HMR quanto a ocupação do gargalo da rede. Na sequência, foram adicionados 4 e 8 pares, introduzindo desta forma uma progressão exponencial do número de conexões TCP atualizadas sendo

estabelecidas. Os testes realizados contemplam as variações de configuração do TCP (detalhes podem ser verificados na Tabela 5).

4.4.3.1 Tempo de Execução



(a) Tempo de execução do HMR. (b) Pacotes perdidos.

Figura 19 – Tempo de execução do HMR e perda de pacotes variando o número de pares cliente-servidor de tráfego de *background*.

Analisando os resultados visualizados na Figura 19(a) pode ser verificado a variabilidade do tempo de execução do HMR entre as diferentes configurações de marcação de filas (RED1 e RED2) e cenários de algoritmos de controle de congestionamento. Para o cenário atualizado(DC) com o gerenciamento RED1 o tempo de execução do HMR é uniforme entre as execuções com 2, 4 e 8 pares concorrentes. Avançando a análise para o cenário legado(MV) as execuções onde foram introduzidas 2 e 4 pares de tráfego de *background*, apresentam para o HMR um tempo de execução similar ao cenário atualizado(DC). No entanto, com 8 pares, o tempo de execução da aplicação HMR sofre um aumento considerável, deixando claro o quanto o número de conexões TCP sendo estabelecidas compartilhando os mesmos recursos, pode prejudicar a tempo de resposta de aplicações com algoritmo TCP desatualizados (no caso, HMR).

Em ambientes virtualizados (VCC), com poucas conexões a virtualização mantém um tempo de execução similar ao ambiente atualizado, bem como no ambiente legado. Entretanto, com 4 pares de fluxos concorrentes em execução, ocorreu um aumento significativo. Porém, aumentando para 8 o número de pares, o aumento no tempo de execução foi na ordem de 10 vezes mais, ou seja um aumento aproximado de 1000%.

Com a configuração de gerenciamento de fila RED2, no cenário com ECN, ocorreu uma grande variabilidade nos resultados. Sobretudo, no cenário com TCPs

heterogêneos (legados e atualizados) uma menor variabilidade foi observada quando comparada com o cenário atualizado. Com VCC ativada, para 8 pares de tráfego de *background*, o resultado mostrou que a virtualização do controle de congestionamento para a aplicação HMR não é vantajosa com maior número de conexões.

Como conclusão, a configuração RED2 apresentou um tempo de execução maior que RED1 em todos as variações de configuração do TCP para 2 e 4 pares concorrentes. Porém, com 8 pares de *background* o maior tempo de execução foi verificado na configuração RED1. É fato que o limite inferior de marcação de fila e um intervalo mais aberto em RED2, proporcionam maior número de pacotes marcados.

4.4.3.2 Perda de Pacotes

A Figura 19(b) apresenta as perdas de pacotes em todos os cenários em que o experimento foi executado. Inicialmente, os resultados no cenário atualizado(DC) para a configuração de gerenciamento de gerenciamento de fila RED1 são analisados. Observa-se que a perda de pacotes é praticamente a mesma com todos os 3 conjuntos de pares introduzidos, e tem um valor bem abaixo do apresentado para o cenário legado(MV). No cenário legado(MV) houve um aumento significativo na ocorrência de perdas mostrando que o número de conexões TCP estabelecidas influencia na perda de pacotes e consequentemente afeta a vazão da rede. Com o aumento exponencial do número de conexões sendo estabelecidas, também ocorre um aumento exponencial da perdas de pacotes. Cada conexão concorrente estabelecida gerencia a abertura e fechamento de sua janela de dados se baseando na informação de congestionamento recebida do núcleo da rede para evitar perdas de pacotes. Por outro lado, penalizam o HMR que neste cenário sofre intensa perda de pacotes quando o fluxo entra na fila no *switch* em um instante onde os limites de gerenciamento da fila são superados.

No cenário virtualizado(VCC), não ocorre uma redução no número de pacotes perdidos nas execuções com 2 e 4 pares. Porém, observa-se que com 8 pares houve um aumento significativo nas perdas em relação ao cenário legado(MV). A partir dos resultados apresentados é possível observar que a virtualização para o HMR piora o *goodput* quando esta está competindo com muitas conexões com algoritmo TCP atualizados.

Analisando os resultados com configuração de gerenciamento de fila RED2 verifica-se que o cenário atualizado(DC) não sofre alterações significativas quando novas conexões são inseridas. Neste caso todas as conexões reduzem o envio baseadas na quantidade de pacotes com *bit* ECN ativado recebidos do núcleo da rede, antes que ocorra perda de pacotes. No entanto no cenário legado(MV), quando o HMR é penalizado pela condição desatualizado do controle de congestionamento que

é percebido quando ACKs duplicados são recebidos ou na ocorrência de *timeouts*, o número de pacotes perdidos tem um aumento significativo. Nesta configuração de gerenciamento de fila, a diferença entre a perda de pacotes entre o cenário legado(MV) e o cenário com a VCC ativada é praticamente a mesma.

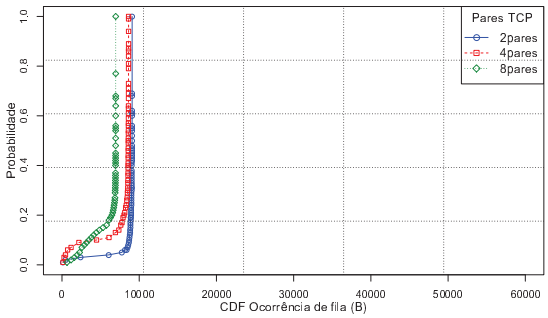
De uma maneira geral a configuração de gerenciamento de fila RED2 apresenta uma maior ocorrências de perdas em relação à configuração RED1, com exceção da execução com 8 pares onde a virtualização com RED1 se mostrou mais desvantajosa até mesmo em relação a um cenário heterogêneo. Embora o intervalo aberto dos limites de ocupação da fila definido para marcar a possível ocorrência de congestionamento, inicie aos 30000 *bytes* e finalize aos 90000 *bytes*, ela não consegue evitar a ocorrência de timeouts das conexões não otimizadas.

4.4.3.3 Formação de Filas

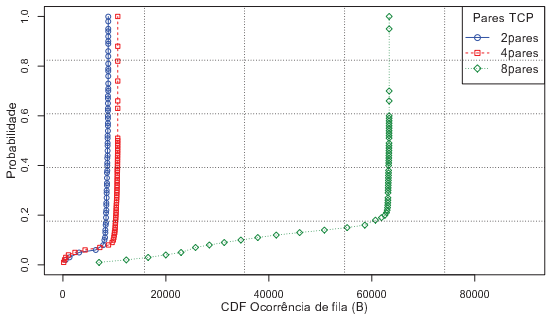
Na configuração de gerenciamento de fila RED1, no cenário atualizado conforme Figura 20(a), a probabilidade de formação de filas se mostrou inferior a 10000 *bytes*. Uma peculiaridade apresentada neste cenário foi a inversão do número de filas entre a execução com 2 pares que se mostrou maior que a ocorrência de formação de filas apresentado na execução com 8 pares. Isso se explica pela agressividade do algoritmo TCP atualizado em preencher a janela de envio de dados. Com mais conexões recebendo orientações baseadas em marcação ECN, como acontece com a execução com 8 pares, cada conexão gerencia o crescimento de sua janela de dados, que acontece mais lentamente, em função da quantidade de conexões compartilhando o meio, diminuindo assim a formação de fila.

No cenário legado(MV), Figura 20(b), a execução com 2 e 4 pares mantém em um patamar de ocupação de fila próximo ao cenário atualizado(DC), porém com 8 pares a formação de filas é superior chegando a uma probabilidade de mais de 60000 *bytes* de ocupação do *buffer*. Ou seja, quando a aplicação HMR é executada em uma configuração que não utiliza ECN competindo com poucos fluxos que nativamente reconhecem ECN, o impacto na formação e filas não á significativo. Porém a medida que mais destes fluxos são inseridos, a probabilidade de aumento na ocorrência de filas aumenta significativamente.

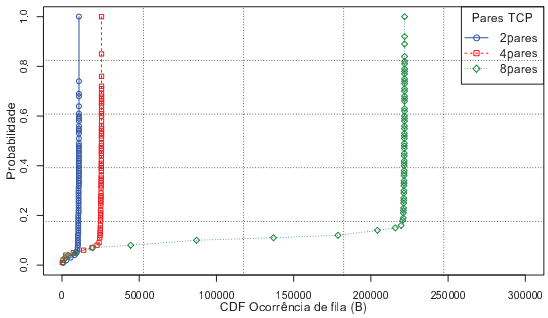
Ainda, com configurações RED1, no cenário com VCC ativada, Figura 20(c), o comportamento da formação de filas segue uma dinâmica bem parecida com o cenário legado(MV) apresentado na Figura 20(b). A formação de filas está diretamente relacionada ao número de conexões sendo estabelecidas. Porém uma ressalva a ser considerada é que aumentou o número de *bytes* na fila, por exemplo, com 8 pares a formação da fila superou a quantidade de 230000 *bytes*, acima dos 60000 para a mesma execução com cenário legado(MV). Mesmo em execuções com 2 e 4 pares a



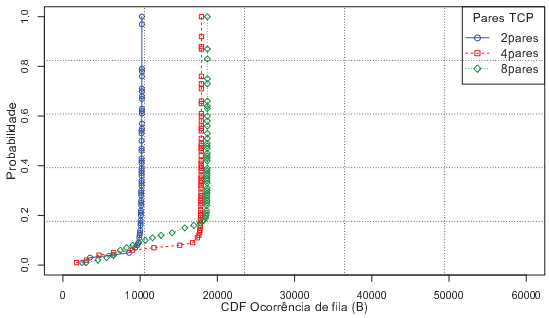
(a) Com ECN - RED1.



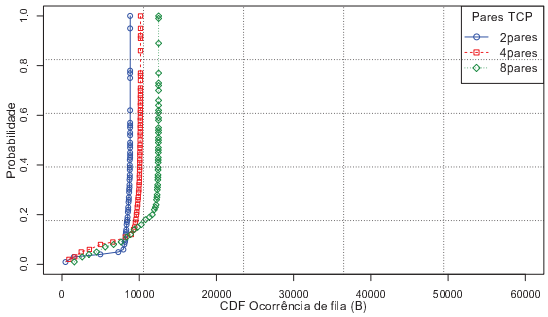
(b) Sem ECN - RED1.



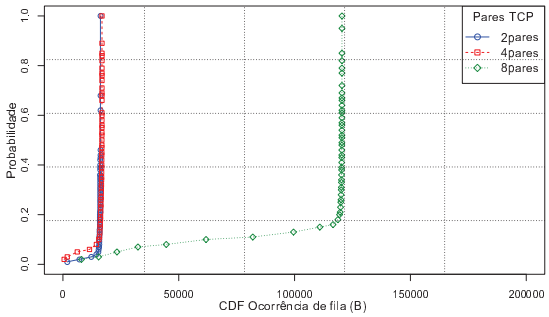
(c) Virtualizado - RED1.



(d) Com ECN - RED2.



(e) Sem ECN - RED2.



(f) Virtualizado - RED2.

Figura 20 – Formação de filas variando o número de pares executando tráfego de *background*.

ocupação da fila de mostrou superior ao apresentado no cenário heterogêneo(MV). O que pode-se concluir com os resultados apresentado é que a VCC não consegue melhorar a taxa de ocupação das filas em relação ao ambiente heterogêneo(MV), ao contrário, consegue um aumento na ocupação do *buffer*, prejudicando diretamente as aplicações HMR.

Na configuração de fila RED2, em cenário atualizado(DC), conforme a Fi-

gura 20(d), a formação de fila ocorre de forma similar ao observado nos demais experimentos, sendo fortemente influenciada pelo número de conexões. Com 2 pares a formação de fila se mantém em torno de 10000 *bytes*, porém, ao introduzir mais pares, a quantidade de *bytes* ocupando o *buffer* praticamente é dobrada. Interessante observar que a ocupação com 4 pares é levemente menor que a registrada com 8 pares, devido a estabilização da janela de dados.

Surpreendentemente nesta configuração de gerenciamento de fila, o cenário legado(MV) apresentou uma probabilidade de ocupação da fila menor que no cenário atualizado(DC), demonstrando um variabilidade muito pequena entre os pares executados, Figura 20(e). Na seção 4.4.3.4 é discutido a razão de tal resultado. Com o cenário virtualizado(DC) na Figura 20(f), a variabilidade tem um comportamento bem diferente: com 2 e 4 pares mantém uma ocupação de próxima do 25000 *bytes* e com 8 pares eleva essa ocupação para aproximadamente 125000 *bytes* mostrando um aumento de 400%.

4.4.3.4 Principais Observações

O propósito deste experimento foi quantificar a ocupação da fila sem qualificar os fluxos que a originam. Não é possível identificar quantos destes *bytes* na fila pertencem a uma aplicação que use ECN ou quais são de aplicações HMR sem consciência de marcação de pacotes que extrapolam o limite da fila. Entretanto, conforme indicado pela Tabela 8, dado o volume de *background* trafegado, e as características de comunicação de aplicações HMR (discutidas na Seção 2.3.1), é possível concluir que a VCC piora a ocupação do *buffer* quando HMR está competindo com mais fluxos ECN, independente da configuração de gerenciamento de fila, seja ela RED1 ou RED2.

4.5 DISCUSSÃO E PRINCIPAIS OBSERVAÇÕES

Iniciamos as observações dos resultados do experimento 1, pelo tempo de execução do HMR com RED1 em cenário atualizado, legado e virtualizado representados pelas Figuras 13(a), 13(b) e 13(c). Na sequência as Figuras 13(d), 13(e) e 13(f) representam os resultados da configuração RED2. Com fluxos menores nota-se que RED1 apresenta uma maior variabilidade do tempo de execução. No cenário sem ECN ambas configurações tiveram uma grande variabilidade com cargas acima de 8GB, demonstrando que a carga de *background* consegue introduzir uma perturbação no desempenho da aplicação *HMR*. Já no cenário com VCC ativada, em cargas maiores o RED2 teve um resultado pior do que a outra configuração. Concluímos que a maior perda de pacotes obtida na configuração RED2 constatada na Seção 4.4.1.2

Métrica	Observação
Tempo de execução	- Com 8 pares sem ECN ocorreu um aumento significativo no tempo de execução de HMR; - A virtualização com 8 pares piorou o resultado em relação ao resultado sem ECN com RED1; - A configuração RED1 apresentou melhor resultado que RED2 com 2 e 4 pares, ao passo que foi pior com 8 pares.
Pacotes perdidos	- De uma maneira geral a configuração RED2 apresentou maior perda que RED1; - As maiores perdas ocorreram em execuções sem ECN e com VCC; - As perdas crescem exponencialmente conforme novas conexões são estabelecidas.
Formação de filas	- Com 8 pares a ocupação da fila em cenário virtualizado tem um crescimento exponencial.

Tabela 8 – Principais observações no experimento 3.

se reflete no aumento do tempo de execução em função da retransmissão de pacotes perdidos.

As perdas de pacotes, neste experimento, estão representadas nas Figuras 14(a), 14(b) e 14(c), respectivamente para os cenários, atualizado (com ECN), legado (sem ECN) e atualizado (com VCC ativada), para a configuração de gerenciamento de fila RED1. Enquanto que as Figuras 14(d), 14(e) e 14(f) representam os respectivos cenários na configuração de fila RED2. Observa-se que em todos os cenários a configuração RED2 apresentou uma perda de pacotes maior do que a configuração RED1, ou seja foi menos eficiente na predição de congestionamento. Quanto a formação de filas (Figuras 15(a), 15(b), 15(c), 15(d), 15(e) e 15(f)), pode-se constatar em todos os cenários a predominância de uma maior ocorrência de filas na configuração RED2. No experimento 2 são utilizados 4 pares de conexões concorrentes que utilizam nativamente ECN. Os testes iniciam com uma carga de 2MB para cada par e a cada rodada de testes a carga é aumentada exponencialmente até o último teste que é realizado com uma carga de 32GB. Analisando a relação entre a perda de pacotes, Figura 17(a), e o tempo de execução, Figura 16(a) em um cenário atualizado, observa-se a existência de uma grande variabilidade na execução. No entanto, pode ser verificado que a mediana oscilou entre 2000 e 3500 pacotes perdidos, enquanto o valor do segundo *quartil* para o tempo de execução oscilou entre 45 e 55 segundos. Observando ainda a Figura 18(a), que representa a formação de fila no cenário atualizado neste experimento, pode ser verificado que as filas foram aumentando a medida que uma carga maior foi aplicada. A maior formação de fila aconteceu com maior carga, ou seja 32GB, cujo valor não ultrapassou 10000 bytes. Uma conclusão assertiva, derivada desta análise é que como a aplicação HMR bem como os demais

fluxos por ela compartilhada, conseguem frear o envio de dados evitando grandes perdas. Em cargas menores ocorreu uma maior perda, em função do ajuste da janela de dados. No entanto com cargas maiores, as conexões estabilizaram o crescimento da janela e atuam em fase de prevenção de congestionamento oscilando a janela na proporção dos pacotes marcados, diminuindo a ocorrência de perdas. As perdas que ocorreram são resultado da ocorrência de *timeout* e casos de rajada onde houve um estouro do limite máximo de marcação do RED.

Em um cenário legado, conforme a Figura 17(b), observa-se um padrão de perda de pacotes nas cargas com 2MB até 512MB oscilando entre 2600 e 3200 pacotes. Semelhante padrão pode ser observado na Figura 16(b), mostrando que o tempo de execução com essas cargas se mantém abaixo de 50 segundos. Com cargas maiores a perda de pacotes apresentou uma variabilidade maior, com execuções apresentando mediana acima de 4000 pacote.. Essa oscilação é esperada em cenários heterogêneos, uma vez que, a ocupação da fila varia conforme a carga, embora não de forma linear, com pequenas inversões como com cargas onde com 32GB apresenta um volume menor que com 2GB Figura 18(b). As aplicações com ECN marcam pacotes que se posicionam na fila entre o parâmetro *min* e o *max* da configuração do RED. Somente descartam pacotes que superam esses limites. Estas conexões tem a oportunidade de reduzir a janela de envio de dados na proporção dos pacotes marcados, o que não acontece com as aplicações legadas que descartam o pacote assim que ele ultrapassa o limite configurado para a fila. Neste momento de redução da janela de dados para aplicações nativas, as aplicações legadas podem ser beneficiadas e apresentar uma vazão momentânea maior, razão pela qual acontece grandes variações nas perdas de pacotes. Ainda na Figura 16(b) verifica-se a grande variabilidade no tempo de execução com maiores cargas concorrentes, em razão da necessidade de reenvio dos pacotes perdidos.

Na Figura 17(c) observa-se que a VCC conseguiu uma redução na perda de pacotes para execuções introduzindo cargas de 2MB até 512MB. Porém com cargas maiores observa-se que houve um aumento das perdas comparadas com o cenário legado, Figura 17(b). Relacionado a isso pode se observar que o tempo de execução se manteve praticamente o mesmo até 512MB, abaixo de 50 segundos. Com cargas maiores a mediana do tempo de execução aumentou consideravelmente e demonstrou uma variabilidade ainda maior com cargas acima de 4GB. A conclusão obtida com esses resultados deixa claro que com cargas pequenas a virtualização consegue melhorar a ocorrência de perdas e consequentemente o tempo de execução, no entanto com cargas maiores a VCC piora os resultados das métricas abordadas.

Em uma análise do experimento 3 onde novas conexões são inseridas a cada rodada de testes, levando em conta as diferenças de perdas de pacotes entre as

configurações de fila RED1 e RED2 é possível constatar que com RED2 ocorrem mais perdas de pacotes que RED1. Nas duas configurações a VCC não conseguiu diminuir a ocorrência de perdas de pacotes, em alguns casos até contribuiu para um aumento, como visto na execução virtualizada com 8 pares, conforme demonstra a Figura 19(b).

Analisando a Figura 19(a), constata-se que o RED2 em cenário legado consegue um tempo de execução maior que com RED1, ou seja um resultado pior. Porém, em cenário legado e virtualizado o RED2 conseguiu menor tempo de execução que com RED1. Com destaque para a execução com 8 pares que com RED1 apresentou um valor elevado. Analisando a relação entre a perda de pacotes e o tempo de execução, no cenário atualizado, é clara a relação em RED1 onde apresenta um número de perdas de pacotes menor em relação aos outros cenários, assim como o tempo de execução mostra-se também bem abaixo dos registrados nas demais execuções. Porém, RED2 apresenta uma aumento no tempo de execução em relação à RED1 embora tenha apresentado ocorrência de perdas semelhantes. No cenário legado as duas configurações apresentam uma perda de pacotes maior que o cenário atualizado. Isso pode ser explicado, porque estamos tratando do tempo de execução da aplicação HMR, no entanto, com relação as pacotes perdidos, não foi feito uma diferenciação se o pacote pertence à aplicação de background, que usa ECN, ou se pertence à aplicação HMR, que não utiliza ECN para evitar a perda de pacotes. Surpreendentemente RED2 conseguiu um menor tempo de execução apesar se ter experimentado uma perda significativa. Já a configuração RED1 elevou o tempo de execução em testes realizados introduzindo 8 pares. No cenário virtualizado a configuração RED1 mostra o maior números de perdas e um maior tempo de execução.

Essa relação mostra que a perda de pacotes causa alterações no tempo de execução, pois pacotes que foram perdidos, precisam ser reenviados para que os dados possam ser entregues completos ao destinatário. A variabilidade apresentada na relação entre as perdas de pacotes e o tempo de execução pode ser entendida quando se pensa que o tempo de execução é especifico da aplicação HMR enquanto que, não foi caracterizado a qual aplicação pertence os pacotes perdidos. Quando uma aplicação perde pacotes a janela de envio que sofre diminuição de volume enviado por estar atuando em modo de prevenção de congestionamento e a janela que sofreu a perda e não das demais conexões.

Ainda no terceiro experimento, com relação a formação de filas em cenário sem ECN a configuração RED2, Figura 20(e), apresenta uma menor formação de fila no *buffer* do *switch* especialmente com mais conexões nativas ECN compartilhando o meio comparada com a configuração RED1 apresentada na Figura 20(b). Em cenário virtualizado(VCC) o RED1, Figura 20(c) também mostra um volume maior de filas comparado com RED2 Figura 20(f). Porém no cenário atualizado(DC) o melhor

resultado de ocorrência de filas é obtido no gerenciamento de fila RED1.

4.6 CONSIDERAÇÕES FINAIS

A Tabela 9 resume a caracterização e apresenta um diagnóstico do uso do VCC em aplicações HMR, baseado nas observações parciais apresentadas nas Tabelas 6, 7 e 8.

Experimento	**VCC recomendado**	**VCC não recomendado**
Experimento 1	até 8GB	acima de 8GB
Experimento 2	até 1GB	acima de 2GB
Experimento 3	até 4 pares	acima de 4 pares para RED1

Tabela 9 – Recomendações de uso do VCC para aplicações HMR.

A virtualização do controle de congestionamento foi apontada como uma solução para atuar na discrepância entre diversos algoritmos TCP. Na análise inicial das propostas, os autores realizaram estudos considerando tráfego de carga constante, concluindo que a solução é aplicável. A presente análise experimental demonstrou que VCC não é eficiente para HMR em diversos casos. Ou seja, a caracterização apresentada mostra pontos de sobrecarga para uma aplicação real. Sobretudo, a ineficiência é herdada das configurações de marcação em *switches*, realizadas através do RED.

Especificamente, conforme demonstrado na Tabela 6 a VCC é recomendada para cargas pequenas. Para um ambiente com fluxo competidor superior a 8GB, a VCC ao invés de proporcionar um menor de tempo de execução consegue piorar o resultado, ou seja, aumentar o tempo de execução em relação ao cenário com aplicações heterogêneas (sem ECN).

No experimento 2 a VCC apresentou um pior resultado na execução com carga de 1GB para RED1 e a partir de 2GB para RED2. Neste experimento existem 4 pares de aplicação consciente de ECN executando tráfego de *background*, logo o número de conexões manipulando o crescimento da janela é maior que o experimento anterior, bem como o volume de dados. Neste caso podemos ver que a configuração RED2 conseguiu manter a VCC favorável por mais tempo que a configuração RED1.

Analisando as observações parciais do experimento 3, a Tabela 8, demonstra que a virtualização sofre um aumento significativo no tempo de execução quando mais pares concorrentes são introduzidos e competem pelo uso da rede, usando a configuração de gerenciamento de fila RED1, no entanto, com a configuração RED2 não houve diferenças significativas na métrica analisado entre o cenário com VCC e o cenário heterogêneo.

A conclusão que pode se obter a partir dos resultados é que a VCC é fortemente influenciada pela carga de dados que a aplicação sem ECN está injetando na rede, assim como pode ser influenciado pela quantidade de conexões sendo estabelecidas. Ainda, de acordo com os resultados apresentados, após a realização dos experimentos, foi possível constatar a desvantagem das aplicações não otimizadas em relação às aplicações otimizadas.

Os resultados demostram que a virtualização conseguiu melhorar o desempenho das aplicações não otimizadas em ambientes heterogêneos, aproximando os resultados aos obtidos em um ambiente somente com aplicações otimizadas, quando o volume de dados está abaixo de 8GB, conforme evidenciado no experimento 1 Seção 4.4.1.1. Da mesma forma, com cargas até 1GB no experimento 2, Seção 4.4.2.1 que injetou na rede 4 conexões de fluxo de *background* atualizado. Ainda, no experimento 3, Seção 4.4.3.1, pode constatar que em execuções com 8 pares executando aplicações atualizadas, a VCC piorou o resultado em relação ao resultado com cenário homogêneo.

5 CONSIDERAÇÕES E PERSPECTIVAS

A existência de aplicações com diferentes algoritmos de controle de congestionamento é uma realidade em redes de *data center* (DC) de provedores Infraestruturas como Serviço (IaaS). Para garantir receita, os provedores precisam atender a demanda, tanto de aplicações com *Transmission Control Protocol* (TCP) otimizado, quanto de aplicações com TCP não otimizado.

Não é possível executar aplicações com TCP legado (não otimizado) em sistema operacionals (SOs) modernos para garantir uma uniformidade do controle de congestionamento no DC, em função da dependência dessas aplicações de bibliotecas do sistema operacional (SO) em que foram originalmente desenvolvidas. Para garantir equidade para fluxos de aplicações com TCP não otimizado e com TCP otimizado é necessário estabelecer uma forma não invasiva de gerenciamento da comunicação. A virtualização garante que diferentes fluxos tenham o mesmo tipo de gerenciamento do controle de congestionamento enquanto estiverem trafegando no ambiente de rede do DC sem que seja feito qualquer alteração na pilha TCP nas extremidades da comunicação (Emissor e Receptor). Inicialmente, a proposta da Virtualização do Controle de Congestionamento (VCC) indicou que ao ativar o hipervisor para fazer a tradução dos algoritmos dos equipamentos comunicantes para uma versão exclusiva do DC, é possível obter um tempo de execução semelhante ao da execução somente com dispositivos com algoritmos de controle de congestionamento otimizados.

Os resultados obtidos com os experimentos do presente trabalho complementam os trabalhos que originalmente propuseram a VCC (CRONKITE-RATCLIFF et al., 2016; HE et al., 2016) analisando a aplicabilidade com uma aplicação real, o *Hadoop MapReduce* (HMR), baseado na reprodução de comunicações oriundas de um DC real (NEVES; ROSE; KATRINIS, 2015). Os resultados obtidos com os experimentos realizados neste trabalham demonstraram que a aplicação HMR é fortemente influenciada pela carga de dados e quantidade de conexões atualizadas compartilhando a rede.

5.1 SUGESTÕES DE TRABALHOS FUTUROS

Os experimentos apresentados neste trabalho foram executadas usando duas configurações de gerenciamento de fila: RED1 e RED2, cujos parâmetros foram configurado de forma estática. Uma proposição de trabalho futuro é utilizar mecanismo de ajustes dos parâmetros de configuração de fila de forma dinâmica (LU YIFEI FAN, 2018).

Outra proposição é recuperar *logs* reais de aplicações, ativar a VCC e utilizar a ferramenta *TCPReplay* (TCPREPLAY, 2018) para reproduzir o dados em fluxo e analisar o comportamento do tráfego. Assim, outras aplicações podem ser analisadas e a aplicabilidade de VCC estudada. Por fim, analisar a dinâmica de ampliação e redução da janela de dados, bem como o descarte de pacotes e formação de fila de cada configuração de fluxos, com ECN e sem ECN, separadamente durante o processo de virtualização.

5.2 PUBLICAÇÕES REALIZADAS

Durante a pesquisa foram publicados dois artigos em eventos:

(i) Artigo publicado e apresentado no evento Escola Regional de Alto Desempenho (ERAD) 2018,realizado na Universidade Federal do Rio Grande do Sul (UFRGS) no Campus de Porto Alegre. O titulo do artigo é *Desempenho do Hadoop MapReduce sobre um data center com virtualização do controle de congestionamento*.

(ii) Artigo publicado e apresentado no evento Simpósio de Sistemas Computacionais de Alto Desempenho (WSCAD) 2018, com o titulo *Análise da Virtualização do Controle de Congestionamento na Execução de Aplicações Hadoop MapReduce*.

REFERÊNCIAS

AL-FARES, M.; LOUKISSAS, A.; VAHDAT, A. A scalable, commodity data center network architecture. **SIGCOMM Comput. Commun. Rev.**, ACM, v. 38, n. 4, p. 63–74, ago. 2008. ISSN 0146-4833.

AL-FARES, M. et al. Hedera: Dynamic flow scheduling for data center networks. In: **Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation**. Berkeley, CA, USA: USENIX Association, 2010. (NSDI'10), p. 19–19. Disponível em: <http://dl.acm.org/citation.cfm?id=1855711.1855730>.

ALIZADEH, M.; EDSALL, T. On the data path performance of leaf-spine datacenter fabrics. In: **Proceedings of the 2013 IEEE 21st Annual Symposium on High-Performance Interconnects**. Washington, DC, USA: IEEE Computer Society, 2013. (HOTI '13), p. 71–74. ISBN 978-0-7695-5103-6. Disponível em: <http://dx.doi.org/10.1109/HOTI.2013.23>.

ALIZADEH, M. et al. Data center tcp (dctcp). **SIGCOMM Comput. Commun. Rev.**, ACM, v. 41, n. 4, p. –, ago. 2010. ISSN 0146-4833.

ALIZADEH, M.; JAVANMARD, A.; PRABHAKAR, B. Analysis of dctcp: Stability, convergence, and fairness. In: **Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems**. New York, NY, USA: ACM, 2011. (SIGMETRICS '11), p. 73–84. ISBN 978-1-4503-0814-4. Disponível em: <http://doi.acm.org/10.1145/1993744.1993753>.

AVRO. 2018. Disponível em: <http://avro.apache.org/>. Acessado em 10/04/2018.

BAI, W. et al. Enabling ecn in multi-service multi-queue data centers. In: **Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation**. Berkeley, CA, USA: USENIX Association, 2016. (NSDI'16), p. 537–549. ISBN 978-1-931971-29-4. Disponível em: <http://dl.acm.org/citation.cfm?id=2930611.2930646>.

BANI-AHMAD, S.; SA'ADEH, S. Scalability of the dvfs power management technique as applied to 3-tier data center architecture in cloud computing. v. 05, p. 69–93, 01 2017.

BENSON, T.; AKELLA, A.; MALTZ, D. A. Network traffic characteristics of data centers in the wild. In: **Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement**. New York, NY, USA: ACM, 2010. (IMC '10), p. 267–280. ISBN 978-1-4503-0483-2. Disponível em: <http://doi.acm.org/10.1145/1879141.1879175>.

BOXPLOT. 2018. Disponível em: <http://www.abgconsultoria.com.br/blog/boxplot-como-interpretar/>. Acessado em 10/11/2018.

CHIU, D.-M.; JAIN, R. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. **Computer Networks and ISDN Systems**, v. 17, n. 1, p. 1 – 14, 1989. ISSN 0169-7552.

CHOWDHURY, M. et al. Managing data transfers in computer clusters with orchestra. **SIGCOMM Comput. Commun. Rev.**, ACM, v. 41, n. 4, p. 98–109, ago. 2011. ISSN 0146-4833.

CHUKWA. 2018. Disponível em: <http://chukwa.apache.org/>. Acessado em 10/04/2018.

CISCO. **Cisco Data Center Infrastructure: 2.5 Design Guide**. 2007.

CRONKITE-RATCLIFF, B. et al. Virtualized congestion control. In: **Proceedings of the 2016 ACM SIGCOMM Conference**. New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 230–243. ISBN 978-1-4503-4193-6. Disponível em: <http://doi.acm.org/10.1145/2934872.2934889>.

DEAN, J.; GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. **Commun. ACM**, ACM, v. 51, n. 1, p. 107–113, jan. 2008. ISSN 0001-0782.

DOCKER. 2018. Disponível em: <https://www.docker.com//>. Acessado em 10/04/2018.

FILIPOSKA, S.; JUIZ, C. Complex cloud datacenters. **IERI Procedia**, v. 7, p. 8 – 14, 2014. ISSN 2212-6678. International Conference on Applied Computing, Computer Science, and Computer Engineering (ICACC 2013). Disponível em: <http://www.sciencedirect.com/science/article/pii/S2212667814000227>.

FLOYD, S. Tcp and explicit congestion notification. **SIGCOMM Comput. Commun. Rev.**, ACM, v. 24, n. 5, p. 8–23, out. 1994. ISSN 0146-4833.

FLOYD, S.; HENDERSON, T.; GURTOV, A. **The NewReno Modification to TCP's Fast Recovery Algorithm**. United States: RFC Editor, 2004.

GALLARDO, G. A.; BAYNAT, B.; BEGIN, T. Performance modeling of virtual switching systems. In: **2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)**. [S.l.: s.n.], 2016. p. 125–134.

GOLDMAN, A. et al. Capítulo 3 apache hadoop: conceitos teóricos e práticos, evolução e novas possibilidades. 05 2018.

GOVINDAN, R. et al. Evolve or die: High-availability design principles drawn from googles network infrastructure. In: **Proceedings of the 2016 ACM SIGCOMM Conference**. New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 58–72. ISBN 978-1-4503-4193-6. Disponível em: <http://doi.acm.org/10.1145/2934872.2934891>.

GREENBERG, A. et al. Vl2: A scalable and flexible data center network. In: **Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication**. New York, NY, USA: ACM, 2009. (SIGCOMM '09), p. 51–62. ISBN 978-1-60558-594-9. Disponível em: <http://doi.acm.org/10.1145/1592568.1592576>.

GUNARATHNE, T. et al. Mapreduce in the clouds for science. In: **2010 IEEE Second International Conference on Cloud Computing Technology and Science**. [S.l.: s.n.], 2010. p. 565–572.

GUO, C. et al. Bcube: A high performance, server-centric network architecture for modular data centers. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 39, n. 4, p. 63–74, ago. 2009. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1594977.1592577>.

GUO, C. et al. Dcell: A scalable and fault-tolerant network structure for data centers. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 38, n. 4, p. 75–86, ago. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1402946.1402968>.

HA, S.; RHEE, I.; XU, L. Cubic: A new tcp-friendly high-speed tcp variant. **SIGOPS Oper. Syst. Rev.**, ACM, v. 42, n. 5, p. 64–74, jul. 2008. ISSN 0163-5980.

HADOOP. 2018. Disponível em: <https://wiki.apache.org/hadoop/PoweredBy/>. Acessado em 10/04/2018.

HAN, B. et al. Network function virtualization: Challenges and opportunities for innovations. **IEEE Communications Magazine**, v. 53, n. 2, p. 90–97, Feb 2015. ISSN 0163-6804.

HANDIGOL, N. et al. Reproducible network experiments using container-based emulation. In: **Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies**. New York, NY, USA: ACM, 2012. (CoNEXT '12), p. 253–264. ISBN 978-1-4503-1775-7. Disponível em: <http://doi.acm.org/10.1145/2413176.2413206>.

HASEGAWA, G.; MURATA, M. **Survey on Fairness Issues in TCP Congestion Control Mechanisms**. 2001.

HBASE. 2018. Disponível em: <http://hbase.apache.org/>. Acessado em 10/04/2018.

HE, K. et al. Ac/dc tcp: Virtual congestion control enforcement for datacenter networks. In: **Proceedings of the 2016 ACM SIGCOMM Conference**. New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 244–257. ISBN 978-1-4503-4193-6. Disponível em: <http://doi.acm.org/10.1145/2934872.2934903>.

HIVE. 2018. Disponível em: <http://hive.apache.org/>. Acessado em 10/04/2018.

HOFMANN, P.; WOODS, D. Cloud computing: The limits of public clouds for business applications. **IEEE Internet Computing**, v. 14, n. 6, p. 90–93, Nov 2010. ISSN 1089-7801.

HUA, W.; GONG, J. Analysis of tcp bic congestion control implementation. p. 781–784, 08 2012.

IBM. 2018. Disponível em: <https://www-01.ibm.com/software/th/data/bigdata/enterprise.html>. Acessado em 10/04/2018.

JACOBSON, V. Congestion avoidance and control. **SIGCOMM Comput. Commun. Rev.**, ACM, v. 18, n. 4, p. 314–329, ago. 1988. ISSN 0146-4833.

JUDD, G. Attaining the promise and avoiding the pitfalls of tcp in the datacenter. In: **Proc. of the 12th Conf. on Networked Systems Design and Implementation**. Berkeley, CA, USA: USENIX, 2015. (NSDI'15), p. 145–157. ISBN 978-1-931971-218.

KATAL, A.; WAZID, M.; GOUDAR, R. Big data: Issues, challenges, tools and good practices. p. 404–409, 08 2013.

KOCHE, J. C. **Fundamento de Metodologia Científica**. São Paulo: Vozes, 1997.

KOUBA, Z.; TOMANEK, O.; KENCL, L. Evaluation of datacenter network topology influence on hadoop mapreduce performance. **2016 5th IEEE International Conference on Cloud Networking (Cloudnet)**, p. 95–100, 2016.

KüHLEWIND, M.; NEUNER, S.; TRAMMELL, B. On the state of ecn and tcp options on the internet. In: **Proc. of the 14th Int. Conf. on Passive and Active Measurement**. Berlin, Heidelberg: Springer-Verlag, 2013. (PAM'13), p. 135–144. ISBN 978-3-642-36515-7.

KUROSE, J. F.; ROSS, K. W. **Redes de Computadores e a Internet: Uma abordagem top-down**. Trad. 3 ed. São Paulo: Addison Wesley, 2006.

KUZMANOVIC, A. The power of explicit congestion notification. **SIGCOMM Comput. Commun. Rev.**, ACM, v. 35, n. 4, p. 61–72, ago. 2005. ISSN 0146-4833.

KUZMANOVIC, A. et al. **RFC 5562: Adding Explicit Congestion Notification (ECN) Capability to TCPs SYN/ACK Packets**. IETF, 2009.

KVM. 2018. Disponível em: <https://www.linux-kvm.org/page/Main_Page/>. Acessado em 10/04/2018.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: Rapid prototyping for software-defined networks. In: **Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks**. New York, NY, USA: ACM, 2010. (Hotnets-IX), p. 19:1–19:6. ISBN 978-1-4503-0409-2. Disponível em: <http://doi.acm.org/10.1145/1868447.1868466>.

LU YIFEI FAN, X. . Q. L. Dynamic ecn marking threshold algorithm for tcp congestion control in data center networks. Computer Communications, jul. 2018. Disponível em: <https://doi.org/10.1016/j.comcom.2018.07.036>.

LUO, Z. et al. Research of a vpn secure networking model. In: **Proceedings of 2013 2nd International Conference on Measurement, Information and Control**. [S.l.: s.n.], 2013. v. 01, p. 567–569.

MAHJOUB, M. et al. A comparative study of the current cloud computing technologies and offers. In: **Proceedings of the 2011 First International Symposium on Network Cloud Computing and Applications**. Washington, DC, USA: IEEE Computer Society, 2011. (NCCA '11), p. 131–134. ISBN 978-0-7695-4550-9. Disponível em: <https://doi.org/10.1109/NCCA.2011.28>.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>.

MELL, P. M.; GRANCE, T. **SP 800-145. The NIST Definition of Cloud Computing**. Gaithersburg, MD, United States, 2011.

MICROSOFT. **Microsoft Hyper-V**. 2018. Disponível em: <https://docs.microsoft.com/pt-br/virtualization/>. Acessado em 10/04/2018.

MITTAL, R. et al. Timely: Rtt-based congestion control for the datacenter. **SIGCOMM Comput. Commun. Rev.**, ACM, v. 45, n. 4, p. 537–550, ago. 2015. ISSN 0146-4833.

NEVES, M. V.; ROSE, C. A. F. D.; KATRINIS, K. Mremu: An emulation-based framework for datacenter network experimentation using realistic mapreduce traffic. **2015 IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems**, p. 174–177, 2015.

NOORMOHAMMADPOUR, M.; RAGHAVENDRA, C. S. Datacenter traffic control: Understanding techniques and trade-offs. **IEEE Communications Surveys Tutorials**, PP, n. 99, p. 1–1, 2017.

PFAFF, B. et al. The design and implementation of open vswitch. In: **12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)**. Oakland, CA: USENIX Association, 2015. p. 117–130. ISBN 978-1-931971-218.

PIG. 2018. Disponível em: <http://pig.apache.org/>. Acessado em 10/04/2018.

POPA, L. et al. Faircloud: Sharing the network in cloud computing. In: **Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication**. New York, NY, USA: ACM, 2012. (SIGCOMM '12), p. 187–198. ISBN 978-1-4503-1419-0. Disponível em: <http://doi.acm.org/10.1145/2342356.2342396>.

PRIMET, P. V.-B.; ANHALT, F.; KOSLOVSKI, G. Exploring the virtual infrastructure service concept in Grid'5000. In: **20th ITC Specialist Seminar on Network Virtualization**. Hoi An, Vietnam: [s.n.], 2009. Disponível em: <https://hal.inria.fr/hal-00690399>.

RISTA DALVAN GRIEBLER, C. A. F. M. L. G. F. C. Improving the network performance of a container-based cloud environment for hadoop systems. 2017.

ROSENBERG, J.; MATEOS, A. **The Cloud at Your Service**. 1st. ed. Greenwich, CT, USA: Manning Publications Co., 2010. ISBN 1935182528, 9781935182528.

ROY, A. et al. Inside the social network's (datacenter) network. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 45, n. 4, p. 123–137, ago. 2015. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/2829988.2787472>.

SINGLA, A. et al. Jellyfish: Networking data centers randomly. In: **Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)**. San Jose, CA: USENIX, 2012. p. 225–238. ISBN 978-931971-92-8. Disponível em: <https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/singla>.

SOUZA, F. R. de et al. QoS-Aware Virtual Infrastructures Allocation on SDN-based Clouds. In: **Proc. of the Int. Symp. on Cluster, Cloud and Grid Computing**. Madrid, Spain: IEEE, 2017. (CCGRID).

SOUZA, G. S. de et al. Simulation and analysis applied on virtualization to build hadoop clusters. **2015 10th Iberian Conference on Information Systems and Technologies (CISTI)**, p. 1–7, 2015.

STEVENS, W. **TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms**. United States: RFC Editor, 1997.

TCPREPLAY. 2018. Disponível em: <https://tcpreplay.appneta.com//>. Acessado em 08/11/2018.

THUSOO, A. et al. Hive - a petabyte scale data warehouse using hadoop. In: LI, F. et al. (Ed.). **ICDE**. IEEE, 2010. p. 996–1005. ISBN 978-1-4244-5444-0. Disponível em: <http://infolab.stanford.edu/~ragho/hive-icde2010.pdf>.

VMWARE-ESXI. 2018. Disponível em: <https://www.vmware.com/br/products/esxi-and-esx.html/>. Acessado em 10/04/2018.

VSWITCH. 2018. Disponível em: <https://www.vmware.com/support/ws55/doc/ws_net_component_vswitch.html//>. Acessado em 10/04/2018.

WANG, G. et al. A simulation approach to evaluating design decisions in mapreduce setups. In: **2009 IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems**. [S.l.: s.n.], 2009. p. 1–11. ISSN 1526-7539.

WU, H. et al. Ictcp: Incast congestion control for tcp in data-center networks. **IEEE/ACM Transactions on Networking**, v. 21, n. 2, p. 345–358, April 2013. ISSN 1063-6692.

WU, H. et al. Tuning ecn for data center networks. In: **Proc. of the 8th Int. Conf. on Emerging Networking Experiments and Technologies**. [S.l.]: ACM, 2012. (CoNEXT '12), p. 25–36. ISBN 978-1-4503-1775-7.

XEN. 2018. Disponível em: <https://www.citrix.com.br/products/xenserver//>. Acessado em 10/04/2018.

YEDDER, H. B. et al. Comparison of virtualization algorithms and topologies for data center networks. In: **2017 26th International Conference on Computer Communication and Networks (ICCCN)**. [S.l.: s.n.], 2017. p. 1–6.

ZAHAVI, E. et al. Links as a service (laas): Guaranteed tenant isolation in the shared cloud. In: **Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems**. New York, NY, USA: ACM, 2016. (ANCS '16), p. 87–98. ISBN 978-1-4503-4183-7. Disponível em: <http://doi.acm.org/10.1145/2881025.2881028>.

ZHANG, J.; LU, X.; PANDA, D. Performance characterization of hypervisor-and container-based virtualization for hpc on sr-iov enabled infiniband clusters. In: . [S.l.: s.n.], 2016. p. 1777–1784.

ZHOU, J.; MA, Y. Topology discovery algorithm for ethernet networks with incomplete information based on vlan. In: **2016 IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC)**. [S.l.: s.n.], 2016. p. 396–400.

ZOOKEEPER. 2018. Disponível em: <http://zookeeper.apache.org/>. Acessado em 10/04/2018.