Universidade Federal do Espírito Santo

Alextian Bartholomeu Liberato

RDNA: Arquitetura Definida por Resíduos para Redes de *Data Centers*

Vitória-ES
2018

Alextian Bartholomeu Liberato

RDNA: Arquitetura Definida por Resíduos para Redes de *Data Centers*

Tese de doutorado submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como parte dos requisitos exigidos para a obtenção do título de doutor em Ciência da Computação.

Aprovada em 24/08/2018.

Banca examinadora:

Prof. Dr. Magnos Martinello
Orientador

Prof. Dr. Moisés Renato Nunes Ribeiro
Coorientador

Prof. Dr. Christian Rodolfo Esteve Rothenberg
Examinador Externo

Prof. Dr. Leobino Nascimento Sampaio
Examinador Externo

Prof. Dr. Vinícius Fernandes Soares Mota
Examinador Interno

Prof. Dr. Rodolfo da Silva Villaça
Examinador Interno

Dedico esta tese à minha família e amigos. De modo muito especial, dedico este trabalho à minha amada esposa Michelle, e aos meus amados filhos Gabriel e Pedro Henrique. Estendo a dedicatória desta tese aos meus pais Ademar (in memoriam) e Auziliadora.

Agradecimentos

A DEUS; onipotente e bom senhor, agradeço pela vida, paz, saúde, força e amor.

Agradeço de modo especial ao meu orientador prof. Magnos Martinello, ser humano simples, humilde, inteligente e trabalhador, um amigo de coração, um exemplo de caráter, companheirismo e dedicação. Obrigado pela paciência, por acreditar na minha capacidade, pelas palavras de incentivo, pelos momentos de turismo, pela sabedoria, pela experiência de vida compartilhada e por tantas e tantas horas dedicadas, meu muito obrigado de coração.

Ao meu coorientador prof. Moisés Ribeiro, meu respeito e admiração, pelo ser humano incontentável, inquieto e genial, obrigado pelos *e-mails* com artigos, pelas recomendações de livros, filmes e vídeos, obrigado pela ponte Jardim da Penha - *Ballsbridge*, e pela paciência sempre demonstrada, sinto-me privilegiado por conviver com um profissional do seu nível.

A minha esposa pelo apoio incondicional em todos os momentos desta longa caminhada. Aos meus tesouros, meus filhotes obrigado por todas as alegrias, por compreender a ausência constante do papai, cada: não posso, não tenho tempo, outra hora, outro dia, depois, machucava meu coração, mas tenham certeza que isso só fez aumentar a vontade de estar mais perto de vocês meus amores.

A minha família por entender minha ausência constante, de modo especial minha mamãe e meus irmãos Marcio e Tatiane.

Aos professores do departamento de Informática da UFES, Orivaldo e Crediné, pela oportunidade em 2012, Celso pela entrevista e indicação em 2013. Aproveito para agradecer de modo muito especial, ao Rodolfo, sua organização, sabedoria e dedicação é uma inspiração para iniciantes como eu. Roberta, obrigado pelo compartilhamento cultural, pelas rigorosas revisões de texto nos artigos e pela colaboração contínua deste aprendiz.

Aos colegas do LabNerds, saibam que cada conversa, reunião, discussão, artigos e projetos colaboraram para esta tese e que eu sou eternamente grato pela ajuda de todos. De modo especial quero agradecer Diego Mafioletti, Eros, Cristina, Gilmar, Rafinha, Rodolfo Ribeiro e Dione, sem o convívio com vocês certamente eu encontraria mais dificuldades nesta caminhada.

Aos colegas do HPN, Arash e Kote pela colaboração na implementação e testes com as NetFPGAs em Bristol-UK, meu muito obrigado.

Aos colegas do Connect, Francisco, Diarmuid, Fadhil, Danny, Nima, Yi, João, Erika, Pedro, Monica e Farrukh pelo companheirismo e por toda ajuda fornecida. De modo muito especial Professor Luiz pela oportunidade e ao Maicon pela ajuda com o Iris.

Aos colegas do Ifes, Júlio Nardi, Victório e Bruno, pelas conversas e pelo compartilhamento de suas experiências.

À UFES, PPGI e Labtel, que colaboraram com o fornecimento de equipamentos, infraestrutura, material e local para a realização desta tese. De modo especial quero agradecer à colega Glaydis e ao servidor Caio por toda ajuda na secretaria do PPGI.

À CAPES pelo apoio financeiro com o pagamento da bolsa de auxílio durante 35 meses e ao CNPq/FAPES/RNP pelo apoio financeiro aos projetos de pesquisa que permitiu à aquisição dos equipamentos para montar o *Data Center* no LabNerds.

Finalmente agradeço a todos que, de alguma forma, contribuíram com esta tese, incluindo os membros da banca de qualificação pelas valiosas contribuições.

Resumo

Recentemente, temos observado o crescente uso das tecnologias de informação e da comunicação. Instituições e usuários simplesmente necessitam de alta qualidade na conectividade de seus dados, com expectativa de acesso instantâneo a qualquer hora e em qualquer lugar. Um elemento essencial para garantir qualidade na conectividade da nuvem é a arquitetura da rede de comunicação no *Data Center* (DCNs - *Data Center Networks*). Isso ocorre porque uma parte significativa do tráfego da Internet é baseada na comunicação de dados e no processamento que acontece dentro da infraestrutura do *Data Center* (DC). No entanto, os protocolos de roteamento, a forma de encaminhamento e gerenciamento que são executados atualmente, se revelam insuficientes para atender as demandas atuais por conectividade na nuvem. Isto ocorre principalmente pela dependência da operação de busca nas tabelas de encaminhamento, levando à um incremento de latência fim a fim, ademais, mecanismos de recuperação tradicionais utilizam estados adicionais nas tabelas, aumentando a complexidade nas rotinas de gerenciamento, além de reduzir drasticamente a escalabilidade de proteção nas rotas. Outra dificuldade é a comunicação multicast dentro do DC, as soluções existentes são complexas de implementar e não suportam a configuração dos grupos nas taxas atuais requeridas.

Neste contexto, essa tese explora o sistema numérico de resíduos centrado no Teorema Chinês do Resto (TCR) como fundamento, aplicado no projeto de um novo sistema de roteamento para DCN. Mais especificamente, introduzimos a arquitetura RDNA que avança o estado da arte a partir de uma simplificação do modelo de encaminhamento para o núcleo, baseado em uma operação de resíduo (resto da divisão). Nesse sentido, a rota é definida como resíduo entre um identificador de rota e identificadores locais (números primos) atribuídos aos *switches* de núcleo. Os *switches* de borda, recebem entradas configurando os fluxos de acordo com a política de rede definida pelo controlador. Cada fluxo é mapeado na borda, através de um identificador de rota principal e um emergencial. Essas operações de resíduos permitem encaminhar os pacotes pela respectiva porta de saída. Em situações de falha, o identificador de rota emergencial viabiliza rápida recuperação enviando os pacotes por uma porta de saída alternativa.

A RDNA é escalável assumindo uma topologia *2-tier Clos Network* amplamente utilizada em DCNs. Com o objetivo de confrontar a RDNA com outros trabalhos da literatura, analisamos a escalabilidade em termos de número de bits necessário para comunicação unicast e multicast. Na análise, variou-se o número de nós na rede, o grau dos nós e o número de *hosts* físicos para cada topologia. Na comunicação unicast, a RDNA reduziu em 4.5 vezes o tamanho do cabeçalho, comparada à proposta COXCast. Na comunicação multicast, um modelo de programação linear foi concebido para minimizar uma função polinomial. A RDNA reduziu em até 50% o tamanho do cabeçalho comparando com a mesma quantidade de membros por grupo.

Como prova de conceito, dois protótipos foram implementados, um no ambiente emulado Mininet e outro na plataforma NetFPGA SUME. Os resultados mostraram que a RDNA alcança latência determinística no encaminhamento dos pacotes, 600 nanosegundos no tempo de comutação por elemento de núcleo, recuperação de falha ultra-rápida na ordem de microssegundos e sem variação de latência (*jitter*) no núcleo da rede.

Palavras-chave: *Redes de Data Centers*, Arquitetura de Redes, Resiliência, Multicast.

Abstract

Recently, we have seen the increasing use of information and communication technologies. Institutions and users simply require high-quality connectivity of their data, expecting instant access anytime, anywhere. An essential element for providing quality in the connectivity is the architecture of the communication network in Data Center Networks (DCNs). This is because a significant part of Internet traffic is based on data communication and processing that takes place within the Data Center (DC) infrastructure. However, the routing protocols, the forwarding model, and management that are currently running, prove to be insufficient to meet the current demands for cloud connectivity. This is mainly due to the dependency on the table lookup operation, that leads to an end-to-end latency increment. Besides, traditional recovery mechanisms have used additional states in the switch tables, increasing the complexity of management routines, and drastically reducing the scalability for routes protection. Another difficulty is the multicast communication within DC, existing solutions are complex to implement and do not support group configuration at the current required rates.

In this context, this thesis explores the numerical system of residues centered in the Chinese remainder theorem (CRT) as a foundation, applied in the design of a new routing system for DCN. More specifically, we introduce RDNA architecture that advances the state-of-the-art from a simplification of the forwarding model to the core, based on the remainder of the division (modulo). In this sense, the route is defined as a residue between a route identification and local identification (prime numbers) assigned to the core switches. Edge switches receive inputs by configuring flows according to the network policy defined by the controller. Each flow is mapped to the edge, through a primary and an emergency route identification. These residue operations allow forwarding the packet through the respective output port. In failure situations, the emergency route identification enables fast recovery by sending the packets through an alternate output port.

RDNA is scalable by assuming a 2-tier Clos Network topology widely used in DCNs. In order to compare RDNA with other works of the literature, we analyzed the scalability in terms of the number of bits required for unicast and multicast communication. In the analysis, the number of nodes in the network, the degree of the nodes and the number of physical hosts for each topology were varied. In unicast communication, the RDNA reduced by 4.5 times the header size, compared to the COXCast proposal. In multicast communication, a linear programming model is designed to minimize a polynomial function. RDNA reduced header size by up to 50% compared to the same number of members per group.

As proof of concept, two prototypes were implemented, one in the Mininet emulated environment and another in the NetFPGA SUME platform. The results presented that RDNA achieves deterministic latency in packet forwarding, 600 nanoseconds in switching time per core element, ultra-fast failure recovery in the order of microseconds and no latency variation (no jitter) in the core network.

Keywords: Data Centers, Network Architecture, Resilience, Multicast.

Lista de Figuras

2.1 Exemplo de Topologia Fat-tree Fonte: [Niranjan Mysore et al. 2009]. . . . 30
2.2 Exemplo de topologia Hipercubo de grau 4. . . . 30
2.3 Ciclo de recuperação. Fonte: [Vasseur et al. 2004]. . . . 37
2.4 Ciclo de reversão. Fonte: [Vasseur et al. 2004]. . . . 39

3.1 Projeto da arquitetura RDNA. . . . 45
3.2 Sistema de roteamento RDNA para comunicação unicast. . . . 51
3.3 Roteamento RDNA unicast: rápida reação a falha. . . . 52
3.4 Exemplo do funcionamento do algoritmo para geração do IRE com proteção completa da rota principal. Em **(a)** remove-se as arestas do caminho principal gerando um grafo G'. Em **(b)** ilustra-se os pesos dos enlaces entre os *switches* do núcleo. Em **(c)** o menor caminho é encontrado. Em **(d)** o menor caminho entre (S_2,dst), cuja união ($\cup$) resulta no Identificador de Rota Emergencial (IRE). . . . 55
3.5 RDNA multicast com roteamento na origem utilizando resíduos e uma expressão polinomial algébrica específica sobre uma topologia 2-tier Clos Network. . . . 57

4.1 Análise do conjunto de números primos para a configuração $Spine = 08$, $Leaf = 16$, com $portas = 96$. . . . 66
4.2 Cabeçalho multicast requerido, variando o número de portas para configuração 2-tier Clos Network. . . . 68
4.2.1 *multicast RDNA* 68
4.2.2 *COXcast.* 68

5.1 Visão geral da implementação da arquitetura considerando o controlador RDNA, e os *switches* de borda e núcleo. Em nossa arquitetura, um ou mais *switches* de borda podem ser conectados com redes de outros domínios (*gateway*). . . . 70
5.2 Representação dos identificadores de rota mapeado no quadro Ethernet 802.3. 73
5.3 Arquitetura do Switch de núcleo utilizando plataforma NetFPGA SUME 10Gbps. . . . 76
5.4 Configuração topológica do *testbed* com os *switches* de núcleo NetFPGA SUME 10Gbps, além dos geradores de tráfego utilizados. . . . 77
5.5 Formato do Cabeçalho para os experimentos com as FPGAs. . . . 77
5.6 Análise de vazão do TCP em Mbps para o tempo de recuperação com diferentes técnicas e intervalo de confiança de 95%. . . . 79
5.7 Comparação do tempo de recuperação Totalmente proativo versus RDNA. 80

5.8 Topologia Full mesh com 04 (quatro) *switches* de núcleo: (a) sem falha (3 saltos); (b) falha no enlace entre S_5 e S_7 (3 saltos); e (c) falha no enlace entre S_7 e S_{11} (4 saltos). 81
5.9 *Screenshot* das telas no analisador de tráfego com as medidas de latência do *switch* de núcleo (sem falha). 82
5.9.1 *01 (um) salto.* 82
5.9.2 *02 (dois) saltos.* 82
5.10 Medida de latência dos *switches* de núcleo. 83
5.10.1*Sem falha.* 83
5.10.2*Com falha.* 83
5.11 Fluxo dos pacotes através do *pipeline* de processamento: **(a)** visão genérica do encaminhamento dos pacotes utilizando SDN via OpenFlow tradicional [ONF 2012]; **(b)** encaminhamento dos pacotes nos *switches* de núcleo RDNA. 84
5.12 Latência de encaminhamento variando o tamanho dos pacotes. Arquitetura RDNA versus SDN OpenFlow. 85

Lista de Tabelas

4.1 Tamanho do cabeçalho (Bytes) para configurações típicas de um *Data Center* utilizando topologia 2-tier Clos Network. 63
4.2 Sobrecarga (*overhead*) máximo em Bytes versus tamanho da rede. 67

5.1 Exemplo de entrada nas tabelas de fluxo dos *switches* de borda. 74
5.2 Resumo das especificações técnicas dos *switches* avaliados. 85

6.1 Tecnologias e arquiteturas para encaminhamento de pacotes baseado em rótulos. 95
6.2 Tempo médio do processo de recuperação de falha dos protocolos e mecanismos de recuperação. Adaptado de [Cisco 2015]. 101
6.3 Síntese das principais diferenças entre as propostas que utilizam roteamento de origem sob a ótica da proteção de falha no enlace. 103
6.4 Adaptada de [Jia 2014], **G:** # de Grupos ativos; **S:** # de origens; **D:** # de receptores; **I:** # de interfaces, **R:** # de *switches* na árvore, **E:** # de *switches* de borda, **M:** Tamanho dos identificadores, [1] Dependente da Arquitetura do DC, e [2] Dependente do protocolo. 108

Lista de Abreviaturas e Siglas

ASIC *Application-Specific Integrated Circuit*

AWS *Amazon Web Services*

CAPEX *Despesas de Capital*

CSPF *Constrained SPF*

DC *Data Center*

DCN *Redes de Data Center*

DHCP *Dynamic Host Configuration Protocol*

EP *Encoded Path*

FF *Fast Failover*

HPN *High Performance Networks*

IDs *Identificadores*

IETF *Internet Engineering Task Force*

IP *Internet Protocol*

LDP *Label Distribution Protocol*

LER *Label Edge Router*

LSP *Label Switching Path*

LSR *Label Switch Router*

MDC *Máximo Divisor Comum*

MPLS *MultiProtocol Label Switching*

MPLS *Multiprotocol Label Switching*

MSTI *Instância de Árvore de Expansão Múltipla*

MSTP *Multiple Spanning Tree Protocol*

NC *Network Controller*

NIC *Network Interface Card*

OF *OpenFlow Protocol*

OPEX *Despesas Operacionais*

OS *Operating System*

OSPF *Open Shortest Path First*

RSTP *Rapid Spanning Tree Protocol*

RSVP *ReSerVation Protocol*

SDN *Software Defined Networks*

SLA *Service Level Agreement*

SR *Roteamento de Origem*

STP *Spanning Tree Protocol*

TCAM *Ternary Content Addressable Memory*

TCP *Transmission Control Protocol*

TCR *Teorema Chinês do Resto*

TE *Traffic Engineering*

ToR *Top of Rack*

VL2 *Virtual Layer 2*

VLAN *Rede Local Virtual*

VM *Máquina Virtual*

VxLAN *LANs virtuais extensíveis*

Sumário

1 Introdução **12**
- 1.1 Contexto . . . 12
- 1.2 Motivação . . . 15
- 1.3 Hipótese de Trabalho e Proposta de Tese . . . 19
- 1.4 Objetivos Gerais e Específicos . . . 20
- 1.5 Contribuições da Tese . . . 20
- 1.6 Organização da Tese . . . 22

2 Fundamentação Teórica **24**
- 2.1 Teoria dos Números . . . 24
  - 2.1.1 Algoritmo de Euclides . . . 24
  - 2.1.2 Algoritmo de Euclides Estendido . . . 25
  - 2.1.3 Fatoração única . . . 25
  - 2.1.4 Aritmética modular . . . 26
  - 2.1.5 Sistema de congruência . . . 27
  - 2.1.6 Teorema Chinês do Resto . . . 27
- 2.2 Redes de Data Center . . . 28
  - 2.2.1 Centradas em *switches* . . . 29
  - 2.2.2 Centradas em servidores . . . 30
- 2.3 Roteamento em Redes . . . 31
  - 2.3.1 Técnicas para encaminhamento de pacotes . . . 34
- 2.4 Resiliência em Redes . . . 36
  - 2.4.1 Ciclos de recuperação e reversão de falhas . . . 37
  - 2.4.2 Estratégia dos mecanismos de recuperação e suas características . . . 39
- 2.5 Redes Definidas por Software . . . 41
- 2.6 Considerações Finais . . . 43

3 RDNA: Arquitetura Definida por Resíduos para Redes de *Data Centers* **44**
- 3.1 Visão geral . . . 44
- 3.2 Sistema de Roteamento . . . 48
- 3.3 Cálculo do Identificador do *Switch* de Núcleo . . . 49
- 3.4 Codificação do Identificador de Rota para Comunicação Unicast . . . 50
- 3.5 Codificação do Identificador de Rota para Comunicação Unicast Resiliente 51
- 3.6 Formulação de Proteção Completa . . . 54
- 3.7 Codificação RDNA para Comunicação Multicast . . . 56
  - 3.7.1 Calculando o grau da função polinomial . . . 58
  - 3.7.2 Calculando os coeficientes da expressão polinomial . . . 59
- 3.8 Considerações Finais . . . 60

4 Análise de Escalabilidade **62**
4.1 Comunicação Unicast . . . 62
4.2 Comunicação Unicast Resiliente . . . 64
4.3 Comunicação Multicast . . . 65
4.3.1 Impacto sobre o tamanho da rede . . . 65
4.3.2 Impacto sobre o cabeçalho considerando a quantidade de membros do grupo multicast . . . 67
4.4 Considerações Finais . . . 69

5 Prova de Princípios **70**
5.1 Implementação da RDNA . . . 70
5.1.1 Controlador de Rede . . . 71
5.1.2 *Switches* de borda . . . 71
5.1.3 *Switches* de núcleo . . . 73
5.2 RDNA sobre a Plataforma Emulada . . . 74
5.3 RDNA sobre a Plataforma FPGA . . . 76
5.4 Validação . . . 77
5.4.1 Impacto da reação rápida à falha na vazão do TCP . . . 78
5.4.2 Tempo de recuperação da falha . . . 80
5.4.3 Baixa latência e reação rápida à falha . . . 81
5.4.4 Desempenho no encaminhamento de pacotes (RDNA versus SDN OpenFlow tradicional) . . . 83
5.5 Considerações Finais . . . 86

6 Trabalhos Relacionados **88**
6.1 Encaminhamento de Pacotes baseado em Rótulos . . . 88
6.2 Roteamento Resiliente . . . 95
6.2.1 Mecanismos de recuperação reativos e proativos . . . 98
6.2.2 Reação à falha local com roteamento na origem . . . 102
6.3 Comunicação Multicast . . . 105
6.4 Considerações Finais . . . 108

7 Conclusão e Trabalhos Futuros **110**
7.1 Trabalhos Futuros . . . 114

Referências Bibliográficas **117**

Capítulo 1

Introdução

Este capítulo apresenta uma visão geral desta tese, bem como define a base para os capítulos seguintes. Portanto, este capítulo compreende o contexto em que a tese está inserida, a motivação, a hipótese, os objetivos além de descrever a estrutura do documento proposto.

1.1 Contexto

Presenciamos um cenário crescente de popularização e diversificação do uso das tecnologias de informação e comunicação em todos os aspectos da vida humana. Não há atividade da sociedade moderna que não tenha sido afetada pela revolução digital. Desde seu surgimento, novas tecnologias apareceram, ano após ano, criando diferentes produtos e serviços. Podemos dizer que a Internet é uma tecnologia oriunda desta revolução. Ao fazer uma breve retrospectiva da evolução da Internet podemos observar como nossa vida foi diretamente afetada.

A Internet nasceu como uma rede experimental financiada pelo Exército Americano para interconectar grandes universidades e centros de pesquisa. Se dividirmos a história da Internet em fases, teríamos na primeira basicamente a Internet como uma rede que viabilizava o compartilhamento de recursos computacionais de grande e médio porte entre estas instituições. Naturalmente, esta rede foi crescendo e crescendo, mais e mais equipamentos foram se interconectando formando a enorme rede mundial que conhecemos hoje. Atualmente, vivemos a segunda fase, onde temos a Internet como uma rede de pessoas e comunidades.

A necessidade das empresas e pessoas em acessarem seus dados, a qualquer hora e em qualquer lugar, contribuiu para o surgimento de outras tecnologias. Desta necessidade veio um novo paradigma, na qual o uso de serviços locais já não atende ao anseio de seus usuários. Versões *online* de serviços comuns, tais como: armazenamento de arquivos e serviços de impressão, tornaram-se disponíveis a todos os tipos de usuários. Associado a esta mudança de paradigma, a utilização massiva de dispositivos móveis, tais como:

smartphones e *tablets*, faz com que os serviços *online*, sejam cada vez mais demandados pelos usuários, pode-se citar o compartilhamento em fotos, blogs e as mídias sociais em geral [Vassoler 2015]. Este novo paradigma é intitulado Computação em Nuvem, e tem sido usado em vários aspectos da sociedade moderna como negócios, pesquisa, saúde, entretenimento e comércio eletrônico, com um mercado equivalente a 155 bilhões de dólares só em 2014 [Bilal et al. 2014].

A Computação em Nuvem é resultante de uma coleção de recursos computacionais distribuídos, escaláveis, compartilhados e virtualizados. No coração deste ambiente, apto à fornecer infraestrutura, *software*, plataforma, rede e armazenamento em um modelo de serviços na Internet encontram-se os *Data Centers* (DC). Os *Data Centers* constituem a infraestrutura que permite a operação das plataformas de Computação em Nuvem. Tipicamente são construções que abrigam centenas de milhares de servidores físicos e equipamentos de comunicação que estão localizados em um mesmo ambiente devido a exigências ambientais, segurança e manutenção [Bilal et al. 2014]. No núcleo do *Data Center* está localizada as Redes de *Data Center* (DCN - *Data Center Networks*). O planejamento adequado da DCN é crítico, exigindo requisitos de resiliência, desempenho, escalabilidade, eficiência energética e baixo custo [Verdi et al. 2010].

Existem essencialmente duas abordagens de alto nível para a construção de uma DCN. A primeira abordagem utiliza protocolos de comunicação e *hardware* especializados para fornecer interconexão escalável de alto desempenho. Entre as tecnologias mais populares que se enquadram nessa categoria estão InfiniBand [Hamada and Nakasato 2005] e Myrinet [Boden et al. 1995]. A principal desvantagem dessas tecnologias é que elas não são nativamente compatíveis com a pilha de protocolos TCP/IP [LOUKISSAS 2008]. Além do que, exigem interfaces proprietárias de rede para os *hosts* finais, o que eleva muito seu custo.

A segunda abordagem, que visa construir a arquitetura da DCN a partir de equipamentos de prateleira, tendência essa denominada de comoditização (*commoditization*), faz uso principalmente de equipamentos baseados no padrão Ethernet. A motivação para essa abordagem é principalmente econômica, uma vez que a comoditização do DC diminui significativamente o custo total de propriedade (TCO - *Total Cost of Ownership*) [Dell 2010, Singla et al. 2012, Singh et al. 2015]. Dentre as arquiteturas que utilizam o padrão Ethernet, destacam-se as abordagens *switch-only* Fat-tree [Al-Fares et al. 2008], PortLand [Niranjan Mysore et al. 2009] e Leaf-Spine (2-Tiers)[Alizadeh et al. 2014], cujo o encaminhamento de pacotes é feito exclusivamente através de *switches*, em um arranjo organizado em camadas [Al-Fares et al. 2010]. A abordagem em camadas é um fundamento básico dos Data Centers para prover escalabilidade, alto desempenho, flexibilidade, resiliência e facilidade de manutenção [Ramos 2013].

Recentemente, *Data Centers* menores e mais próximos aos usuários, se proliferaram como uma alternativa para o armazenamento e processamento dos dados de maneira efi-

ciente, além de conectar outros aplicativos e usuários que estão distribuídos globalmente. Esse novo modelo, chamado de *Edge Computing*, representa uma tendência com grandes benefícios em relação aos *Data Centers* tradicionais, tais como, menor latência e maior desempenho, redução de custo (CAPEX e OPEX) com uso de equipamentos baseado no padrão Ethernet, rápida implementação, facilidade de gerenciamento e orquestração, dentre outros [Dominicini et al. 2017]. A virtualização[1], a conteinerização [2] e a Internet das Coisas [3], têm levado a um crescimento explosivo no número de pontos finais (*endpoints*) com uma demanda significativa por conectividade da nuvem [Hari et al. 2015].

O aumento da demanda por Computação em Nuvem e o potencial mercado que ela representa, nas mais diversas áreas da atividade humana, impulsionou grandes provedores de acesso público como Amazon, Google e Microsoft a projetar DCs mais eficiente. Um exemplo desse avanço são os serviços especializados para Internet das Coisas, AWS IoT Core, Google IoT Core e Microsoft Azure IoT Hub, respectivamente.

Aplicações de Internet das Coisas buscam melhorar a qualidade de vida das pessoas. Na área da saúde por exemplo, oxímetros de pulso, monitores de glicose, sensores embarcados em equipamentos medicinais entre outros dispositivos transmitirão dados diretamente para médicos e agentes de saúde 24 horas por dia, 7 dias por semana. Até 2030, as áreas urbanas são projetadas para abrigar 60% das pessoas no mundo e uma em cada três pessoas viverá em cidades com pelo menos meio milhão de habitantes. As cidades enfrentarão inúmeras restrições incluindo, a disponibilidade de terras, o crescimento populacional e a gestão dos recursos naturais. Além disso, devem buscar formas de facilitar melhorias no padrão de vida dos cidadãos e visitantes [Open 2018].

Cidades inteligentes deverão melhorar a qualidade de vida das pessoas, através do monitoramento dos níveis de poluição, qualidade do ar no espaço urbano e trânsito, colaborando com a saúde coletiva, economia de energia, segurança, educação, desenvolvimento econômico e outros aspectos do cotidiano. Nas indústrias e empresas a troca de dados entre máquinas podem aumentar a produtividade, reduzir os desperdícios ao longo de toda cadeia produtiva, através de sistemas embarcados que se comunicam entre si, criando novas estratégias de produção. Além disso, a interoperabilidade entre sistemas permitirão melhor interação entre a indústria e os consumidores, a chamada Indústria 4.0 [Wollschlaeger et al. 2017].

[1] Consideramos esse termo como o serviço de compartilhamento da infraestrutura física no DC.

[2] É uma alternativa muito mais leve para a virtualização. Envolve a ação de encapsular um aplicativo em um recipiente com o seu próprio ambiente operacional [Dua et al. 2014].

[3] Este conceito promete tornar qualquer dispositivo eletrônico parte do ambiente da Internet [Buyya and Dastjerdi 2016], com um mercado projetado em 60 trilhões de dólares para os próximos 15 anos [Forbes 2016].

1.2 Motivação

A forma de encaminhamento e gerenciamento que são executados atualmente foram projetados para redes locais e se revelam insuficientes ao longo de uma série de dimensões para atender as demandas por conectividade dentro das Redes de *Data Center* [Martinello et al. 2017]. Basta ver que as soluções tradicionais usam mensagens de controle e sinalização via *broadcast*, dificultando a escalabilidade da rede. Por sua vez, os protocolos de roteamento para redes de intra domínio, que operam em Camada 3, como IGRP [Black 2000] e OSPF [Moy 1998], também impõem suas limitações. O principal desafio está na sobrecarga gerada pelas mensagens de controle e a configuração manual por parte dos administradores, o que pode aumentar consideravelmente o potencial de erros na rede.

Com o objetivo de reduzir a complexidade e flexibilizar o encaminhamento no núcleo das DCNs, diversas propostas defendem o uso do roteamento na origem. São exemplos: o MPLS [Swallow et al. 2005] e Path Switching [Hari et al. 2015]. A parte comum entre estas propostas é a precomputação do caminho para cada destinatário com base em um rótulo (*tag*). Portanto, a indicação do próximo salto não é realizada pelo endereço do destinatário contido no cabeçalho, como nos algoritmos de roteamento tradicionais, mas sim com base em um índice que indica a porta de saída para cada pacote. O uso dos rótulos produz alguns benefícios, como por exemplo, maior facilidade de gerenciamento através da agregação de endereços em classe de serviços e menor tempo de busca nas tabelas. Porém, vai exigir o armazenamento dos estados (índices) nas tabelas de *switches* (*stateful*). Outra desvantagem é que a agregação utilizada no mapeamento dos rótulos se apresenta inadequada para políticas de gerenciamento do fluxo por estado, exigidas nos DCs para atender a crescente demanda por conectividade [Hari et al. 2015].

No que se refere às SDNs (*Software Defined Networks*) aplicadas nos DCs, o modelo de encaminhamento adotado pelos comutadores OpenFlow é baseado em *match-action* [ONF 2012, Xia et al. 2015]. Resumidamente, nesse modelo um subconjunto de *Bytes* (cabeçalho) dos pacotes são buscados em uma tabela, as entradas correspondentes especificam ações que são aplicadas ao pacote determinando o seu encaminhamento. Dessa maneira, equipamentos tradicionais habilitados com OpenFlow fornecem sua tabela de busca (*lookup table*) como operação para comutação de pacotes. Portanto, a manutenção dos estados, o modelo de encaminhamento (*match-action*), e o tempo de busca na tabela, representam consideráveis pontos de variabilidade e incremento da latência de encaminhamento dos pacotes, principalmente pela limitação significativa do espaço da *Ternary Content Addressable Memory* (TCAM), o que exige o armazenamento dos estados em outras memórias mais lentas [Martinello et al. 2017]. Ademais, os mecanismo de recuperação de falha da SDN tradicional requerem o armazenamento de estados adicionais nas tabelas [da Silva et al. 2015], o que gera perda de agilidade na recuperação

de falhas, dificulta o processo de gerenciamento das DCNs. Além disso, para algumas aplicações de IoT em áreas críticas, por exemplo, a latência representa uma restrição crucial e dependendo do tempo de resposta a tomada de decisão pode ser totalmente comprometida.

Na concepção de uma arquitetura SDN canônica, não existe uma distinção clara entre as partes funcionais de uma Rede de *Data Center*, como os elementos do núcleo e da borda, que são fundamentais no projeto de DCNs. De fato, um *switch* habilitado OpenFlow (padrão comum para SDN) está claramente longe de um *design* simples, exigindo suporte para pesquisas em centenas de bits[4] com ações complexas que devem ser especificadas por várias tabelas [Trois et al. 2016]. Além disso, não há desacoplamento entre o roteamento e a política de rede. Quando um controlador SDN decide as funções de um fluxo, também decide o caminho que ele deve passar e os estados de configuração em todos os *switches* intermediários ao longo deste caminho. Ademais, a maioria das implantações de SDN no mundo real são baseadas em sobreposições [Koponen et al. 2014].

A falta de mecanismos de reação rápida à falha dentro das DCNs é outra significante limitação das propostas existentes. Aplicações em áreas críticas dependem de um alto nível de confiabilidade do dado transmitido, pois os mesmos podem ser produzidos apenas em um determinado instante e sua perda ou atraso na entrega compromete significativamente o serviço. Protocolos tradicionais de reação à falha em Camada 2 como *Spanning Tree Protocol* (STP) e suas variações: *Rapid Spanning Tree Protocol* (RSTP) [Marchese and Mongelli 2012] e *Multiple Spanning Tree Protocol* (MSTP) [Marchetti et al. 2005], introduzem nas redes ineficiências substanciais como o bloqueio de alguns caminhos a fim de assegurar que não aconteçam *loops*, além de elevado tempo de convergência. O uso de protocolos de roteamento dinâmicos como o OSPF possui a capacidade de se adaptar na ocorrência de mudanças topológicas. Contudo, o tempo de convergência do OSPF, por exemplo, é extremamente alto, em torno de 40s [Barreto 2008].

Para reduzir a dependência da rede no processo de recuperação pós-falha, algumas arquiteturas como o MPLS (*Multiprotocol Label Switching*) [Rosen et al. 2001] e o Path Switching [Hari et al. 2015], propõem a precomputação de caminhos alternativos para cada destino, de modo que o comutador possa mudar localmente para um caminho alternativo sem esperar pelo processo de convergência no plano de controle. A parte comum dessas arquiteturas é a dependência do armazenamento dos estados nos *switches*, além da necessidade de estados adicionais (*tags/labels*) que permitem contornar a falha, através de caminhos alternativos. No Path Switching esta ação é definida como *fast ReRoute Label*[5].

[4]Um exemplo deste incremento pode ser observado comparando o OpenFlow 1.0, que possui um cabeçalho com 12 (doze) campos, com o OpenFlow 1.4 que possui um cabeçalho com 41 (quarenta e um) campos, um aumento superior a 341% [Bosshart et al. 2014].

[5]Os autores não apresentam detalhes da implementação ou mesmo qualquer resultado desta arquitetura. Basicamente apenas demonstram o potencial da sua proposta para reduzir os estados nos *switches*

No caso do MPLS Fast Reroute [Swallow et al. 2005], ele ainda requer o suporte de um protocolo de sinalização, como *Label Distribution Protocol* (LDP), além dos comutadores MPLS habilitados.

Uma forma de eliminar estados no núcleo da DCN é inserir a informação do caminho (rota) dentro do cabeçalho do pacote, usando o elemento de borda para realizar essa ação. Como eventuais falhas podem ocorrer, também se faz necessário inserir informações que permitam a recuperação a partir do comutador que detectou a falha. Tipicamente esse caminho de *backup* contém trechos (segmentos) que permitem ao mecanismo de recuperação enviar os pacotes por caminhos alternativos. São exemplos destas arquiteturas o Slick Packets [Nguyen et al. 2011] e o SlickFlow [Ramos 2013].

Uma relevante desvantagem, comum das arquiteturas como o Path Switching, MPLS, Slick Packets e SlickFlow está na técnica de encaminhamento utilizada. Em todas se faz necessário a reescrita dos pacotes para a realização do encaminhamento. No MPLS e Path Switching pela necessidade de troca dos *labels* no cabeçalho dos pacotes. Já no Slick Packets e SlickFlow para deslocamento dos bits do caminho codificado no cabeçalho do pacote. A ação de reescrita dos pacotes representa uma tarefa adicional para os *switches* de núcleo/intermediários dentro do DC. Sobretudo, é importante destacar que esses *switches* possuem alta taxa de utilização [Benson et al. 2010], logo, qualquer tarefa extra representa sobrecarga adicional sobre os mesmos, prejudicando o tempo de resposta para aplicações de missão crítica, por exemplo.

Recentes pesquisas aplicadas nas mais diversas áreas tem utilizado um ramo da Teoria do Números que afirma que se conhecemos os restos da divisão euclidiana de um número inteiro a por vários inteiros b, então podemos determinar o restante da divisão de a pelo produto desses inteiros, sob a condição de que os divisores sejam coprimos entre si. O Teorema Chinês do Resto (TCR) tem sido aplicado nas mais diversas áreas do conhecimento, desde segurança da informação com uso em técnicas de codificação e criptografia [Grossschadl 2000], processamento de sinais [Liu and Goutte 2014] e encaminhamento de pacotes [Martinello et al. 2014, Gomes et al. 2016]. De modo especial, no encaminhamento de pacotes esta técnica dispensa a reescrita do pacote e o armazenamento de estados em tabelas dentro dos *switches*, além disso, permite representar a rota completa utilizando apenas um único número inteiro. Essas características podem beneficiar aplicações e serviços que são executados dentro das DCNs, reduzindo o custo na produção de equipamentos núcleo, a latência de encaminhamento, além de oferecer melhor escalabilidade para atender a demanda crescente por conectividade [Hari et al. 2015].

Além da baixa latência e resiliência dentro das DCNs, algumas aplicações requerem comunicação multicast, um exemplo são as aplicações para ambiente de monitoramento, economia de energia, automação, telecomunicações e segurança, ou seja, um determinado dispositivo pode gerar um dado que é transmitido até o DC. Ao entrar na DCN, este

intermediários.

mesmo dado deverá ser enviado para várias aplicações localizadas em diferentes *hosts* físicos, onde diferentes serviços estão sendo executados. Transmissões multicast beneficiam aplicações de grupo, logo devem reduzir o tráfego de rede, economizar custos de comunicação, melhorar a taxa de transferência, fornecendo assim uma infraestrutura eficiente para atender a crescente demanda por conectividade. Essas melhorias são significativas para pequenas redes *Cloud Center* [Jia 2014].

A partir do exposto, pode-se observar a necessidade de definir uma nova arquitetura para redes de *Data Center* que atendam as demandas crescente por conectividade. Portanto, esta nova arquitetura deve ser:

1. Capaz de suportar a explosão de crescimento dos estados de encaminhamento, porém, mantendo a capacidade de definir rotas por micro-fluxos sem a necessidade de armazenamento dos estados em tabelas nos elementos núcleo da rede. Tradicionalmente, esta explosão de estados tem sido resolvida por agregação de fluxos, por exemplo, agregação de prefixos IP ou por tunelamento, com o MPLS [Hari et al. 2015]. A agregação de prefixos de endereços e as técnicas de tunelamento ajudam a controlar o estado de encaminhamento nas redes com melhor esforço. No entanto, são inadequados para gerenciar o estado de roteamento baseado em políticas ou estado de encaminhamento por micro-fluxos.

2. Eficiente na comutação de pacotes para comunicação unicast e multicast, colaborando assim para evolução das aplicações e serviços nas mais diversas áreas e de modo especial para aplicações e serviços que requerem baixa latência de encaminhamento dentro dos DCs;

3. Flexível para suportar a separação clara entre as partes funcionais de uma Rede de *Data Center*, como núcleo, borda e gerenciamento, a fim de facilitar o desenvolvimento contínuo de forma inovadora, independente e escalável;

4. Capaz de fornecer proteção em todo o caminho principal de forma simples, sem reescrita do pacote a cada salto, sem troca de mensagens de controle e sinalização, permitindo que os elementos de núcleo sejam eficientes não apenas na comutação do pacote, mas também ativos no processo de recuperação de falha sem a dependência única de elementos externos.

Diante do cenário apresentado, observou-se que as tecnologias existentes aplicadas a *Data Centers* ainda carecem de melhorias significativas para atender as demandas por conectividade, principalmente para aplicações críticas que dependem de baixa latência e confiabilidade. Percebemos através de pesquisas na literatura e das atividades desenvolvidas pelo grupo de pesquisa (LabNerds), que o Teorema Chinês do Resto possui características relevantes na área de roteamento e encaminhamento, mantendo os *switches*

de núcleo/intermediário sem tabela. Esta característica simplifica o gerenciamento das rotas nas DCNs, além de possivelmente reduzir o CAPEX com uso de equipamentos mais baratos por dispensarem o uso de memórias caras como as TCAMs. Também observamos que o TCR tem sido pouco explorado nas DCNs, tornando-se o principal motivador desta tese.

1.3 Hipótese de Trabalho e Proposta de Tese

A hipótese principal adotada nesta tese é que o Sistema Numérico de Resíduos com base no Teorema Chinês do Resto fornece o fundamento para concepção de uma arquitetura de rede para *Data Center* original, criando um sistema de roteamento centrado em resíduos, permitindo o uso de técnicas de encaminhamento baseada apenas em operações de módulo, além de não requerer tabela nos *switches* de núcleo da rede, facilitando o mapeamento de rotas fim a fim. Portanto, se existir uma infraestrutura de DCN que utilize os benefícios do TCR em *Data Centers* construídos em topologias 2-tier, teremos então uma arquitetura escalável, provendo roteamento resiliente ao oferecer um mecanismo de reação extremamente rápido à falhas ao longo de toda a rota principal, suportando comunicação unicast e multicast, podendo ser implementada reutilizando o enquadramento universal do padrão Ethernet, através da programabilidade dos resíduos nos campos existentes.

Em conformidade com a hipótese assumida nesta tese, propomos explorar os benefícios do TCR e estudar sua aplicação em *Data Centers* com até 1440 servidores físicos, cuja infraestrutura deve estar mais próxima aos usuários do que a típica Computação em Nuvem pública atual. Portanto, o desenvolvimento desse trabalho baseia-se nas seguintes premissas:

- O TCR permite o uso de um modelo de encaminhamento sem tabela baseado em operação de módulo (resíduos) com mapeamento de rota realizado por um elemento de borda. Esta característica vai permitir superar o desafio da escalabilidade no número de estados exigido na conectividade do DC, mantendo os estados dos microfluxos restritos apenas aos elementos de borda;

- O avanço das SDNs fornecem os recursos necessários para uma distinção clara entre os blocos funcionais de borda, núcleo e gerenciamento. Deste modo, podemos aplicar os avanços obtidos com as SDNs em termos de programabilidade da rede a fim de validar nossa investigação.

A seguir descreveremos os objetivos gerais e específicos desta tese.

1.4 Objetivos Gerais e Específicos

Objetivo geral: Propor uma arquitetura para infraestrutura de rede destinada à *Data Centers*, que suporte a comunicação unicast e multicast de forma eficiente, atendendo assim as demandas por conectividade para os serviços que requerem uma combinação de baixa latência de encaminhamento e resiliência dentro dos DCs. Além disso, esta nova arquitetura deverá fornecer latência determinística de encaminhamento mesmo sob condições de falhas transitórias, ser escalável e compatível com o padrão Ethernet, sem garantia de baixa latência fim a fim.

Objetivos específicos:

São objetivos específicos:

1. Demonstrar um sistema de roteamento unicast que auxiliem a redução da latência de encaminhamento e sua variabilidade nas DCNs baseado nos resíduos numéricos;

2. Formular um mecanismo de encaminhamento unicast que permita a rápida recuperação de falhas, seja em dispositivos ou em enlaces físicos, de forma proativa;

3. Desenvolver um sistema de roteamento para comunicação multicast que permita representar de forma eficiente a árvore de multicast sem a necessidade de armazenamento de estados nos *switches* de núcleo e sem modificar o pacote a cada salto.

1.5 Contribuições da Tese

O uso do Teorema Chinês do Resto para suportar um sistema de roteamento centrado em resíduos não é novo, e já foi apresentada pelas propostas: Keyflow [Martinello et al. 2014], COXcast [Jia 2014] e KAR [Gomes et al. 2016]. Entretanto, no decorrer de toda revisão da literatura e ao longo de todo desenvolvimento desta tese, percebemos que o TCR foi pouco explorado pelos muitos trabalhos relacionados à área de *Data Center*, especialmente na comunicação unicast considerando recuperação à falha e na comunicação multicast.

Neste contexto, destacamos duas contribuições principais desta tese: i) projeto, implementação e validação de um mecanismo de proteção que permite reagir rapidamente à falhas ao longo de toda rota principal. Este mecanismo garante total proteção contra falhas em qualquer ponto da rota principal utilizando apenas um único identificador de rota emergencial, mantendo os *switches* de núcleo da topologia sem tabela; ii) projeto e análise de escalabilidade de um mecanismo de roteamento e encaminhamento para comunicação multicast. Nosso mecanismo constrói a árvore de multicast com base na codificação de um mapa de bits que permite representar o conjunto de portas no qual os pacotes devem ser transmitidos, sem reescrita de cabeçalho no pacote a cada salto e sem tabela de encaminhamento.

O mecanismo de recuperação apresentado nesta tese possui características únicas não encontradas na literatura, mesmo sem manter nenhum estado armazenado em tabela e sem depender de nenhum protocolo de convergência, graças a inovadora abordagem projetada, implementada e validada, cada pacote na comunicação unicast que entra na rede de núcleo transporta um identificador emergencial que permite a rápida reação à falha ao longo de toda rota principal. Portanto, a **Arquitetura Definida por Resíduos para Redes de *Data Centers* (RDNA)**, amplia o roteamento na origem para suportar uma abordagem resiliente determinística baseado em nosso mecanismo de proteção. O identificador de rota emergencial é calculado de forma a representar o conjunto de *switches* de núcleo e suas respectivas portas de saída. Para isso, nosso algoritmo de proteção considera rotas disjuntas e a menor distância em número de saltos a partir de cada *switch* de núcleo que compõe a rota principal. Os resultados demonstram que nosso mecanismo de reação à falha mantem a latência de encaminhamento determinística sob condição de não congestionamento das filas.

No que diz respeito ao roteamento e encaminhamento multicast, projetamos uma abordagem única não encontrada na literatura. Nosso esquema utiliza em conjunto duas subáreas da Teoria dos Números, o TCR e os polinômios algébricos. A RDNA monta a árvore de multicast a partir de um mapa de bits (*bitmap*) que representa o conjunto de portas em cada *switch* que faz parte da rota multicast. Um simples algoritmo calcula o grau do polinômio com base nos identificadores localizados nos *switches* de núcleo e na quantidade total de portas. A partir do *bitmap*, um segundo algoritmo calcula os coeficientes utilizados na expressão polinomial específica. Esta transformadora abordagem mantem-se integrada com os benefícios da comunicação unicast apresentadas nesta tese. Ademais, a abordagem RDNA para comunicação multicast é consideravelmente mais escalável que as concorrentes, reduzindo em até 50% o tamanho do cabeçalho multicast para algumas das topologias avaliadas, conforme resultado desta tese.

Em síntese, a arquitetura RDNA oferece consideráveis avanços em termos de redução de latência no mecanismo de encaminhamento unicast. Do mesmo modo, a RDNA fornece um eficiente mecanismo de roteamento e encaminhamento multicast. Além disso, também consideramos a implementação e validação da RDNA como uma valiosa contribuição. No melhor conhecimento deste autor, essa é a primeira implementação de uma proposta SDN que de fato usa uma rede de núcleo *fabric* em netFPGA SUME 10G para redes de *Data Center*, utilizando o TCR como sistema de roteamento e encaminhamento, incluindo reação rápida à falhas na rede de núcleo. Todas as propostas anteriores de rede de núcleo eram puramente conceituais [Casado et al. 2012, Hari et al. 2015].

1.6 Organização da Tese

Este capítulo apresentou a primeira parte desta tese, na qual foram descritos os aspectos gerais incluindo o contexto, a motivação, a hipótese de trabalho, a proposta de tese e os objetivos gerais e específicos.

O restante desta tese está organizado da seguinte forma:

- Capítulo 2: Este capítulo descreve os conceitos necessários para o entendimento da arquitetura proposta. De uma forma resumida e objetiva este capítulo apresenta uma explicação sobre os conceitos e teorias utilizados nesta tese, familiarizando o leitor com os termos mais comuns, como o Teorema Chinês do Resto, redes de *Data Center*, roteamento e resiliência em redes de *Data Center*. Este capítulo é importante, pois existe um considerável quantidade de conceitos relacionados a diferentes áreas do conhecimento.

- Capítulo 3: Este capítulo descreve em detalhes o projeto, arranjo e a composição da arquitetura proposta denominada, **RDNA: Arquitetura Definida por Resíduos para Redes de *Data Centers***. Portanto, explicamos o funcionamento do sistema de roteamento unicast e multicast, incluindo o cálculo dos identificadores de rota principal e emergencial. Ademais, dois novos conceitos são descritos neste capítulo: i) o projeto do mecanismo de reação rápida à falha utilizando um rota emergencial que protege todo o caminho principal; e ii) o sistema de roteamento utilizando o TCR em conjunto com polinômios algébricos específicos que permitem desenhar de forma eficiente a árvore de multicast dentro do núcleo da topologia do DC sem tabela.

- Capítulo 4: Este capítulo apresenta a análise de escalabilidade da arquitetura RDNA. Em linhas gerais os resultados também são comparados com outras propostas que possuem similaridade com a nossa.

- Capítulo 5: Este capítulo descreve os experimentos e os resultados obtidos com o objetivo de validar nossa arquitetura para Redes de *Data Center*. Neste mesmo capítulo também é discutido aspectos relevantes da implementação. Com o objetivo de melhor contextualizar os resultados às necessidades de conectividade da nuvem nos DCs, estruturamos cada experimento em um contexto, facilitando o entendimento para o leitor das contribuições alcançadas com esta tese.

- Capítulo 6: Este capítulo analisa alguns dos trabalhos mais relevantes no contexto desta tese. Uma análise crítica também é apresentada, debatendo as principais limitações das tecnologias atuais aplicadas em redes de *Data Center*. Em especial, discutimos três áreas específicas: i) o encaminhamento de pacotes baseado em rótulos; ii) a recuperação de falhas; e por último, iii) as diferentes abordagens para

comunicação multicast. Neste capítulo também situamos a tese no estado da arte no contexto das DCNs. Devido a pluralidade de conceitos envolvidos optamos em discutir as diferentes propostas sempre comparando com nossa arquitetura, deste modo, o leitor compreenderá os avanços obtidos com este tese.

- Capítulo 7: Finalmente, este capítulo descreve em linhas gerais as considerações finais e conclui a tese, além disso, também descrevemos neste capítulo algumas limitações observadas em nossa arquitetura e os trabalhos futuros, direcionando os próximos passos para esta pesquisa.

Capítulo 2

Fundamentação Teórica

O objetivo desse capítulo é apresentar ao leitor aspectos teóricos relevantes que serão utilizados ao longo desta tese. Nas próximas seções apresentaremos uma visão geral da Teoria dos Números, base para o entendimento do Teorema Chinês do Resto, além dos conceitos de Redes de *Data Center*, roteamento e resiliência. Todos os conceitos aqui apresentados formam o alicerce teórico indispensável para o entendimento da arquitetura RDNA.

2.1 Teoria dos Números

A Teoria dos Números é a parte da matemática dedicada ao estudo dos números inteiros e seus amigos [Martinez et al. 2010]. Portanto, nesta seção, apresentamos os fundamentos matemáticos básicos, os quais são necessários para o entendimento do Teorema Chinês do Resto [Garner 1959, Alves et al. 2011, Boyer and Gomide 2012, Alves Junior 2012, de Souto Filho 2015].

2.1.1 Algoritmo de Euclides

O Algoritmo de Euclides ou também chamado de Algoritmo Euclidiano é descrito como uma das formas de se determinar o máximo divisor comum (MDC) entre dois números inteiros diferentes de zero. Para calcular o MDC entre a e b, pode-se utilizar a sequência de divisões ilustrada na Equação 2.1.

$$\begin{aligned} a &= bq_1 + r_1 \quad e \quad 0 \leq r_1 < b \\ b &= r_1q_2 + r_2 \quad e \quad 0 \leq r_2 < r_1 \\ r_1 &= r_2q_3 + r3 \quad e \quad 0 \leq r_3 < r_2 \\ r_2 &= r_3q_4 + r4 \quad e \quad 0 \leq r_4 < r_3 \end{aligned} \tag{2.1}$$

Observando a sequência de restos, nota-se que o seguinte é sempre menor que o an-

terior, porém todos são maiores ou iguais a zero [Alves Junior 2012]. Logo, pode-se reescrever matematicamente a Equação 2.1 para $b > r_1 > r_2 > r_3 > r_4 \cdots \geq 0$.

2.1.2 Algoritmo de Euclides Estendido

Sejam a e b inteiros positivos e d o MDC desses números, usando a extensão do algoritmo de Euclides [Alves Junior 2012] é possível encontrar inteiros α e β (Equação 2.2).

$$\alpha \cdot a + \beta \cdot b = d \tag{2.2}$$

Portanto, o Algoritmo Euclidiano Estendido além de calcular o máximo divisor comum entre a,$b \in \mathbb{Z}$, fornece simultaneamente os coeficientes α, $\beta \in \mathbb{Z}$, tais que $\alpha \cdot a + \beta \cdot b = MDC(a,b)$. Para efetuar os cálculos correspondentes a uma determinada divisão, basta salvar os dados referentes as duas divisões imediatamente anteriores. A sequência de divisões pode ser obtida conforme Equação 2.3.

$$\begin{aligned}
a &= bq_1 + r_1 \quad e \quad r_1 = ax_1 + by_1 \\
a &= bq_1 + r_1 \quad e \quad r_1 = ax_1 + by_1 \\
b &= r_1q_2 + r_2 \quad e \quad r_2 = ax_2 + by_2 \\
r_1 &= r_2q_3 + r_3 \quad e \quad r_3 = ax_3 + by_3 \\
r_2 &= r_3q_4 + r_4 \quad e \quad r_4 = ax_4 + by_4 \\
&\vdots \\
r_{n-3} &= r_{n-2}q_{n-1} + r_{n-1} \quad e \quad r_{n-1} = ax_{n-1} + by_{n-1} \\
r_{n-2} &= r_{n-1}q_n \quad e \quad r_n = 0
\end{aligned} \tag{2.3}$$

Os números $x_1, \cdots, x_{n-1}$ e $y_1, \cdots, y_{n-1}$ são os inteiros que serão determinados pelo algoritmo, cujo a condição de parada é que os últimos valores encontrados para x e y sejam α e β, ou seja, $\alpha = x_{n-1}$ e $\beta = y_{n-1}$ [Alves Junior 2012].

Encontramos o Algoritmo de Euclides Estendido em especial, aplicado para o cálculo de inverso modular. Se a e b são coprimos, então α é o inverso modular de a módulo b e β é o inverso modular de b módulo a. Essa propriedade é amplamente utilizada no estudo em criptografia [da Silva and Chaves 2004, Coutinho 2005] e segurança de dados [Shishira and Vipin 2015].

2.1.3 Fatoração única

Cada número inteiro pode ser escrito como um produto de números primos. Um número natural é um número primo quando ele tem exatamente dois divisores: o número um e ele mesmo. Conforme apresentado em [Alves et al. 2011, Alves Junior 2012] o teorema

da fatoração única pode ser enunciado da seguinte forma: dado um inteiro positivo $n \geq 2$ pode-se sempre escrevê-lo, de modo único, na forma representada na Equação 2.4.

$$n = P_1^{E_1} \quad \cdots \quad P_K^{E_K}, \tag{2.4}$$

onde $1 < P_1\ P_2\ P_3 \cdots P_K$ são números primos e $E_1 \cdots E_K$ são números inteiros positivos. Os expoentes $E_1 \cdots E_K$ são chamados de multiplicidades. Assim, a multiplicidade de P_1 na fatoração de n é E_1. Observa-se também que n tem K fatores primos distintos, mas que a quantidade total de fatores primos (distintos ou não) é a soma das multiplicidades $E_1 + \cdots + E_K$ [Alves Junior 2012].

Finalmente, observa-se que o teorema de fatoração única demonstra duas propriedades, são elas: i) todo número inteiro pode ser escrito como um produto de números primos e ii) só há uma escolha possível de primos e expoentes para a fatoração de um inteiro desejado [Alves et al. 2011].

2.1.4 Aritmética modular

A definição chave no estudo de sistemas aritméticos modulares estabelece uma relação entre pares de números em relação a um número especial chamado m ou módulo.
Definição: Dois números a e b são congruentes módulo m se diferirem por um inteiro múltiplo de m, ou seja, $b - a = km$ para algum $k \in \mathbb{Z}$. Essa equivalência é escrita como $a \equiv b \pmod{m}$ [Devlin 1998].

Pode-se verificar as propriedades da congruência módulo m. Primeiro, a propriedade reflexiva. Se a é um inteiro. Para demonstrar que $a \equiv a \pmod{m}$, verifica-se, por definição, que a diferença $a - a$ é um múltiplo de m, pois 0 é um múltiplo de qualquer número inteiro. Analisando as propriedade simétrica, verifica-se que se $a \equiv b \pmod{m}$, então $a - b$ é um múltiplo de m. Mas $b - a = -(a - b)$; logo, $b - a$ também é múltiplo de m. Portanto, $b \equiv a \pmod{m}$ [Alves Junior 2012].

Considerando a propriedade transitiva, dado $a \equiv b \pmod{m}$ e $b \equiv c \pmod{m}$, onde a, b e c são inteiros. A primeira congruência mostra que $a - b$ é múltiplo de m; a segunda mostra que $b - c$ é múltiplo de m. Logo, somando múltiplos de m teremos novamente múltiplos de m; portanto, $(a - b) + (b - c) = (a - c)$ que é um múltiplo de m, ou seja, $a \equiv c \pmod{m}$.

[Alves Junior 2012] descreve pela relação de congruência módulo m, o conjunto quociente de $\mathbb{Z}$ tem uma notação própria, $\mathbb{Z}_n$; e seu nome é conjunto dos inteiros módulo m. Seja $a \in \mathbb{Z}$, a classe a é formada pelos $b \in \mathbb{Z}$ que satisfaçam $b - a$ múltiplos de m. Portanto, a classe a pode ser descrita conforme Equação 2.5.

$$\overline{a} = a + km : k, a, b, \in \mathbb{Z} \tag{2.5}$$

Em particular $\overline{0}$ é o conjunto dos múltiplos de m, fazendo com que se $a \in \mathbb{Z}$, então podemos dividi-lo por m, obtendo q e r inteiros, tais que $a = nq + r$ e $0 \leq r \leq n - 1$. Logo, $a - r = mq$ que é um múltiplo de m. Portanto, $a \equiv r \pmod{m}$, demonstra que o conjunto quociente $\mathbb{Z}_n$ é formado pelas classes $\overline{0}$, $\overline{1}$, $\cdots$, $\overline{m-1}$.

2.1.5 Sistema de congruência

Considerando o caso de uma única equação linear $a \equiv b \pmod{m}$, onde m é um inteiro positivo, podemos assumir que essas equações são simples de resolver quando $\overline{a}$ tem inverso em $\mathbb{Z}_n$, se e somente se, a e m são primos entre si. Se a não tem inverso com m, pode-se dizer que $mdc(a,m) \neq 1$. Assim, se a equação $a \equiv b \pmod{m}$ tem solução, isso quer dizer que existem $x, y \in \mathbb{Z}$ tais que $ax - ny = b$ [Alves Junior 2012]. Portanto, somente é permitido se o máximo divisor comum entre a e m divide b. Logo, se a tem inverso em $\mathbb{Z}_n$ teremos esta condição satisfeita, ou seja, $mdc(a,m) = 1$.

2.1.6 Teorema Chinês do Resto

O Teorema Chinês do Resto (TCR) apareceu no livro de Sun Zi, um matemático na China antiga. O livro é conhecido pelo nome, "Manual de aritmética do Sol", de Sun Zi Suanjing. A data exata é desconhecida, mas é razoável para levá-lo para ser durante o primeiro século d.C [Boyer and Gomide 2012].

A aplicação do TCR ocorre em quase todas as áreas da matemática. Como citado por [Boyer and Gomide 2012], a aritmética modular já era estudada desde a antiguidade, em que o algoritmo chinês foi usado na solução prática de problemas relativos a construção de paredes, na base de edifícios, comércio de alimentos, entrega de informações, e cálculo de calendários na antiguidade. Recentemente, observamos sua aplicabilidade está também presente em diversas áreas da computação, especialmente na teoria da informação e codificação, em relação a vários aspectos de algoritmos, criptografia e cálculos modulares [de Souto Filho 2015].

Considere o seguinte sistema com duas Equações demonstrado em 2.6:

$$\begin{aligned} &(1). \quad x \equiv a \pmod{w} \\ &(2). \quad x \equiv b \pmod{m} \end{aligned} \tag{2.6}$$

Seguindo o exemplo ilustrado em [Alves Junior 2012], podemos reescrever na forma $x = a + wy$, onde y é um inteiro qualquer. Assim, podemos substituir x na segunda equação, obtendo $wy \equiv (b - a) \pmod{m}$.

Naturalmente, para que esta equação tenha solução o MDC entre w e m precisa dividir $b - a$, portanto, assume-se que $mdc(m,w) = 1$. Com isso, $\overline{w}$ tem inverso em $\mathbb{Z}_n$. Chamando $\overline{\alpha} \in \mathbb{Z}_n$ o inverso, a solução da Equação 2.6 é $y \equiv \alpha(b - a) \pmod{m}$. Assim,

$y = \alpha(b - a) + nz$, onde z é um número inteiro. Substituindo na equação que dá x em função de y, temos $x = a + w\alpha(b - a) + wmz$.

Portanto, dado que $\overline{m\alpha} = \overline{1}$ em $\mathbb{Z}$, podemos assumir que existe algum número inteiro β tal que $1 - w\alpha = m\beta$. O que nos leva a Equação 2.7.

$$x = a(1 - w\alpha) + wb\alpha + wmz = am\beta + wb\alpha + wmz \tag{2.7}$$

De fato, $1 = w\alpha + m\beta$. Assumindo $mdc(w,m) = 1$, basta aplicar o Algoritmo de Euclides Estendido a w e m para obter α e β. Resumindo: se $mdc(w,m) = 1$, então o sistema demonstrado na Equação 2.7 sempre tem como soluções os números $am\beta + bw\alpha + kwm$, onde k é um inteiro qualquer [Alves Junior 2012].

Uma equação linear pode ter mais de uma solução se o módulo for composto. O sistema ilustrado na Equação 2.7 de fato, possui infinitas soluções quando se trata de soluções inteiras, já que há uma solução para cada k. Diz-se que x e y são dois inteiros que são soluções do sistema descrito acima. Então, tem-se que $x \equiv a \pmod{w}$ e $y \equiv a \pmod{w}$. Como se trata de duas equações com o mesmo módulo, pode-se realizar a subtração entre elas. Obtendo-se $x - y \equiv 0 \pmod{w}$. Assim, w divide $x - y$. Repetindo com a segunda equação, conclui-se que m divide $(x - y)$. Supondo que $mdc(w,m) = 1$, tem-se que wm divide $x - y$. Assim, o sistema possui infinitas soluções inteiras, mas apenas uma solução em $\mathbb{Z}_{wm}$. Portanto, conforme descrito em [Alves Junior 2012], tudo isso pode ser resumido em um teorema: **Teorema Chinês do Resto**: Sejam, w e m inteiros positivos e primos entre si. O sistema da Equação 2.6 sempre tem uma única solução em $\mathbb{Z}_{wm}$.

Este atributo do TCR é utilizado em nossa arquitetura para determinar a porta de saída em nosso mecanismo de encaminhamento e descoberta do valor dos coeficientes em uma expressão polinomial. Maiores detalhes serão apresentados no Capítulo 3.

2.2 Redes de Data Center

O modelo computacional dos *Data Centers* começou com os *mainframes* na década 1960. Posteriormente, após o surgimento dos microcomputadores, uma busca constante por altas capacidades de armazenamento se estabeleceu. Esta fase foi seguida pelos sistemas distribuídos baseados no modelo cliente/servidor e, subsequentemente, pelo crescimento explosivo da Internet e da Web [Verdi et al. 2010].

Mais recentemente, a evolução das técnicas de virtualização tem propiciado o desenvolvimento de aplicações que compartilham a mesma infraestrutura de *hardware*, alavancando o surgimento de soluções para serviços em nuvem [Verdi et al. 2010]. A habilidade de partilhar de maneira orquestrada os mesmos recursos físicos entre diferentes inquilinos [Barham et al. 2003, Staff 2013], favorece economicamente os locatários e as opera-

doras que ofertam esses serviços [Mudigonda et al. 2011, MICROSOFT 2013, NSF 2013]. Essa demanda inicial motivou a criação de mega centros de dados para acolher uma ampla gama de serviços, tais como pesquisa na Web, *e-commerce*, *backup* de armazenamento, *streaming* de vídeo, computação de alto desempenho e análise de dados [Verdi et al. 2010, Armbrust et al. 2010].

Para hospedar esses aplicativos, as redes de *Data Center* precisam ser escaláveis, dinâmicas, eficientes, tolerante à falhas e fáceis de gerenciar [Gill et al. 2011]. A diversidade de aplicações que podem ser hospedadas usando o modelo em nuvem inclui desde aplicações comerciais, sistemas Web tradicionais até aplicações científicas para processamento paralelo e aplicações para dispositivos móveis. Estas diferentes aplicações requerem diferentes arquiteturas, e isto tem motivado a pesquisa e o desenvolvimento de soluções que atendam a esta demanda, no sentido de criar mecanismos escaláveis, com alto desempenho e custos mínimos. Isto inclui a pesquisa em infraestrutura voltada para a eficiência energética, cabeamento, resfriamento e, principalmente, infraestrutura de interconexão dos servidores e de roteamento [Verdi et al. 2010].

Certos desta importância, a comunidade acadêmica tem proposto diversas arquiteturas que fornecem melhor escalabilidade e desempenho em redes de *Data Center*. Algumas propostas utilizam rede centradas em *switches* [Kim et al. 2011, Abu-Libdeh et al. 2010, Al-Fares et al. 2008, Alizadeh et al. 2014]. Outras propostas como: [Guo et al. 2008, Guo et al. 2009, Wu et al. 2009], priorizam o uso de servidores para tomada de decisão de roteamento e encaminhamento.

2.2.1 Centradas em *switches*

As redes de *Data Center* centradas em *switches*, tais como: Fat-Tree [Al-Fares et al. 2008], PortLand [Niranjan Mysore et al. 2009] e Leaf-Spine [Ghorbani et al. 2017] (2-tiers Clos Network), são topologias tradicionais, constituídas por servidores físicos agrupados em armários (*racks*) e interligados através de um *switch* Ethernet de acesso no topo desses armários, comumente chamados de ToR (Top-of-Rack). Tipicamente estas redes utilizam duas ou três camadas de *switches* de alta capacidade. Entretanto, dispositivos como roteadores e balanceadores de carga também são necessários para rotear os pacotes pelos vários segmentos da rede e distribuir a carga entre os elementos de encaminhamento, respectivamente [Al-Fares et al. 2008].

Tradicionalmente é possível observar neste tipo de *Datacenter*, um arranjo topológico formado por uma árvore com múltiplos nós na raiz, conforme ilustrado na Figura 2.1, os *switches* de acesso (*Edge*) conectam os servidores aos *switches* de agregação (*Agregation*), que agregam armários de servidores. Por fim, estes se ligam a um terceiro nível de *switches* que formam o núcleo da rede (*Core*). É por meio desta abordagem que as topologias centradas em redes conseguem escalar em número de máquinas. Outra vantagem neste

tipo de abordagem (Fat-Tree e Clos), é possibilidade de projetar redes com probabilidade zero de bloqueio para qualquer tráfego de entrada [Yuan et al. 2009, Verdi et al. 2010], ou seja, sem *oversubscription*.

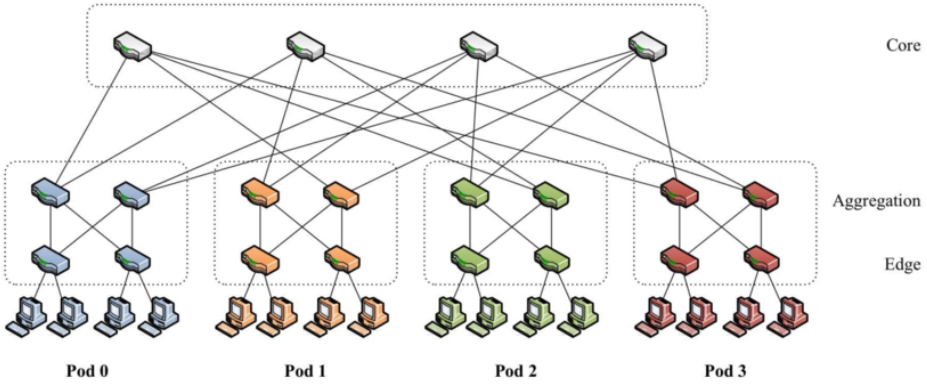


Figura 2.1: Exemplo de Topologia Fat-tree Fonte: [Niranjan Mysore et al. 2009].

2.2.2 Centradas em servidores

Por outro lado, redes centradas em servidores, tais como: DCell [Guo et al. 2008] e BCube [Guo et al. 2009], são topologias que dispensam equipamentos tradicionais de rede, para roteamento, bem como para balanceamento da carga na rede. Todavia, podem utilizar mini-switches com a função de expandir o número de portas dos servidores. Estas topologias mais escaláveis, evidentemente, implicam que os servidores físicos devem possuir mais de uma interface de rede, a fim de realizarem as funções de: roteamento, encaminhamento e balanceamento de carga do tráfego da rede, além do processamento de dados dos usuários.

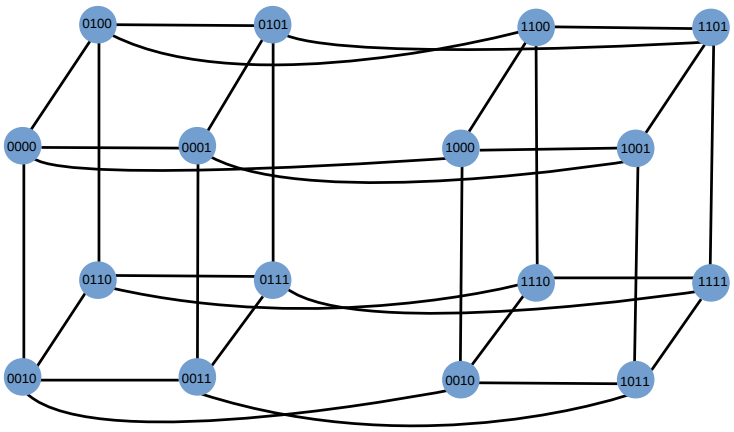


Figura 2.2: Exemplo de topologia Hipercubo de grau 4.

No exemplo apresentado na Figura 2.2, é possível observar uma DCN centrada em servidores como um Hipercubo de grau 4. Note que não existe nenhum elemento tra-

dicional de rede e que cada nó possui 04 (quatro) interfaces de rede para estabelecer as conexões com seus vizinhos. Outras topologias como: MDCube [Wu et al. 2009] e Twin [Vassoler et al. 2014] também utilizam os próprios servidores na função de roteamento e encaminhamento dos pacotes.

Devido a sua habilidade de multiplexar os núcleos da CPU entre as aplicações e os processos de encaminhamento de pacotes [Popa et al. 2010], a abordagem de redes de DC centrado em servidores abre novas possibilidades para testar novos projetos e serviços. Ainda, tal como mostrado em [Greenberg et al. 2008a] esta redução pode alcançar até 15% do custo total de um DC.

Estas características tornam as arquiteturas centradas em servidores uma forma de interconexão atrativa financeiramente. No entanto, um problema muito conhecido das arquiteturas centradas em servidores é a sobrecarga que o encaminhamento por múltiplos saltos gera sobre a CPU dos servidores. Por exemplo, a arquitetura DCell e BCube utilizam em média 40% e 36% de da CPU para encaminhamento do tráfego de trânsito, respectivamente [Vassoler 2015]. Dependendo do cenário de tráfego, o uso de CPU pode atingir valores próximos a 90% [Landau et al. 2011]. Como resultado a latência pode se tornar um fator proibitivo para aplicações sensíveis ao atraso, tais como: bolsas de valores (100 μs a 1μs) e computação de alto desempenho (μs 1 a 10 μs) [Kompella et al. 2009, Vassoler 2015].

Independentemente da arquitetura utilizada no DC, seja centrada em comutadores ou em servidores, uma parte cada vez maior do tráfego da Internet é baseada na comunicação de dados e de processamento que ocorre dentro da infraestrutura do DC (**back-ends**). Estudos realizados por [Sun et al. 2015], revelaram que 80% do tráfego que ocorre dentro da infraestrutura do DC é resultante de requisições internas. Portanto, a taxa de ocupação dos recursos da infraestrutura da rede se mantem alta, próximo a capacidade máxima disponível, durante todo o tempo de operação.

2.3 Roteamento em Redes

Em uma rede de computadores de modo geral, a troca de informações (texto, imagem, música etc) é realizada predominantemente por meio de pequenas unidades de dados. Essas unidades são conhecidas como pacotes. Cabe ao algoritmo de roteamento [Iren et al. 1999] duas funções de igual importância, descritas a seguir.

A primeira função tem como objetivo encontrar o melhor caminho. Este melhor caminho não necessariamente significa ser o caminho mais curto, podendo ser, por exemplo, o caminho com a maior vazão ou menor congestionado ou a combinação destas restrições, entre o nó origem, ou seja, a entidade que está enviando as informações. Por sua vez, o nó destino, é a entidade que irá receber as informações transmitidas. Este processo é conhecido como definição da **rota** entre o par de entidades envolvidas na comunicação.

A segunda função tem como objetivo **encaminhar** de maneira correta os pacotes até o nó destino, ou seja, após a definição da rota, cabe ao mecanismo de encaminhamento o envio dos pacotes recebidos para a(s) porta(s) de saída do nó. Basicamente, a primeira função é executada em um **plano de controle**, já a segunda função ocorre no **plano de dados**.

Na literatura, encontramos diversas classificações para os algoritmos de roteamento. Por exemplo, os algoritmos de roteamento podem determinar rotas baseados em caminho único ou em múltiplos caminhos [Keshav 1997]. No roteamento baseado em caminho único, um nó mantém apenas um caminho para cada destino. No roteamento baseado em múltiplos caminhos, um nó mantém um caminho principal e vários caminhos secundários para um destino. Se o caminho principal estiver indisponível por alguma razão, então um caminho secundário ou alternativo pode ser usado [Barreto 2008].

Também podem ser classificados como roteamento externo e interno, que também são conhecidos como roteamento de intra domínio e inter domínio. O roteamento inter domínio não está no escopo deste trabalho. Informações adicionais sobre roteamento externo podem ser encontradas em [Akashi et al. 2006, Rothenberg et al. 2012, Gupta et al. 2014, Braun and Menth 2014].

No contexto das DCNs, encontramos o uso de algoritmos de roteamento interno, principalmente por se tratar de uma entidade de gerência administrativa única. Um exemplo de algoritmo de roteamento comumente utilizado é o Dijkstra [Dijkstra 1959], implementado no protocolo OSPF [Moy 1998]. Este pode ser empregado para encontrar os caminhos de menor custo entre os *hosts*. O *Open Shortest Path First* (OSPF) é um protocolo que utiliza a troca de informação de rotas observando o estado da rede. Falhas na rede em topologias de grande escala são comuns. O OSPF pode detectar essas falhas e, em seguida, transmitir as informações para todos os *switches/routers* para assim, evitarem enlaces indisponíveis. Para potencializar a capacidade da rede, técnicas de engenharia de tráfego tais como ECMP podem ser utilizadas no DC [Verdi et al. 2010], permitindo que vários caminhos alternativos entre origem/destino sejam usados.

O principal problema conhecido em DCN no contexto do roteamento, está no fato de que as tabelas de roteamento, utilizadas no processo de encaminhamento, crescem na medida que a rede aumenta de tamanho. Logo, manter mais de uma rota entre pares (origem e destino) pode se tornar uma proposta inviável. O roteamento hierárquico oferece significativa redução na quantidade de informações contidas nas tabelas de roteamento. Sua principal característica é o uso de endereços que são dependentes dos nomes [Kleinrock and Kamoun 1977, Cowen 2001, Thorup and Zwick 2001], onde informações sobre a localização dos nós estão incorporadas nos endereços atribuídos a todos os nós. Contudo, o uso de esquema de endereçamento dependente do nome compromete o suporte para mobilidade [Pasquini et al. 2011]. Um exemplo claro deste prejuízo está na movimentação das VMs entre os diversos servidores físicos, resultando na perda de

agilidade no DC.

Para flexibilizar a movimentação, mantendo uma estrutura de roteamento escalável, naturalmente novas propostas surgiram. Ao contrário do roteamento hierárquico estas propostas utilizam mecanismos de encaminhamento que são independentes do nome, também conhecidas como roteamento plano. Tradicionalmente, estas propostas constroem suas tabelas de roteamento utilizando identificadores estáveis, a grande maioria destas propostas independentes de nomes disponíveis na literatura requerem o uso de mecanismos de mapeamento para traduzir os identificadores de plano em localizadores, que indicam a posição atual dos nós, ou seja, os identificadores planos não são efetivamente utilizados para transportar o tráfego através da rede [Pasquini et al. 2011]. Um exemplo desta proposta encontramos na arquitetura VL2, que usa duas famílias de endereços para identificação de *switches* e aplicações. A identificação das aplicações não muda e através de roteamento plano, portanto, a realocação de aplicações em diferentes servidores é facilitada [Verdi et al. 2010].

Outro tipo de roteamento que visa flexibilizar o gerenciamento dos endereços, independentes da topologia e que pode ser utilizado nas DCNs é o *source routing* (SR) ou roteamento explicito. Neste tipo de roteamento o emissor de um pacote determina a sequência completa de nós através dos quais encaminhar o pacote. Ou seja, o emissor enumera explicitamente a rota, tradicionalmente no cabeçalho do pacote, identificando para cada *hop* no caminho do pacote o endereço do próximo nó ao qual o pacote deve ser transmitido [Rosen et al. 2001, Ramos 2013]. Uma característica deste tipo de mecanismo é o encaminhamento dos pacotes com base em informações contidas nos próprios pacotes [Gomes et al. 2016].

Recentemente, o roteamento na origem tem motivado diversas propostas de novas arquitetura, em especial, sempre buscando o equilíbrio entre a inteligência distribuída e a otimização da visão centralizada [Desmouceaux et al. 2017]. O roteamento por segmentos (*segment routing*), tem como principal característica o uso de mecanismos de comutação dos pacotes com base em uma lista ordenada de instruções [Abdelsalam et al. 2017, Filsfils et al. 2014]. Tais instruções permitem realizar políticas de ponta a ponta em qualquer caminho topológico, mantendo o estado por fluxo apenas no nó de origem [Desmouceaux et al. 2018].

Os algoritmos também podem ser classificados como estáticos ou dinâmicos. No roteamento estático a definição da rota é realizada manualmente pelo administrador da rede, determinando estaticamente (roteamento estático) o caminho ou percurso do pacote em cada *switch* ou roteador. O principal desafio nesta abordagem é que, na medida que mais elementos de rede são adicionados, maior a dificuldade na gerência e manutenção desses estados na rede. Além disso, o roteamento estático é vulnerável à falha de recursos, portanto, necessita da atuação obrigatória do administrador de rede. Isso pode levar a uma má utilização dos recursos da rede (normalmente, o mesmo conjunto de enlaces/equipa-

mentos é utilizado, enquanto outros caminhos possíveis são desprezados) [Barreto 2008].

Por sua vez, os algoritmos de roteamento dinâmico possuem mecanismos que permitem a descoberta automática das rotas. Isto é possível, graças a troca de informações (pacotes de controle) entre os elementos da rede. *Switches* e roteadores adjacentes podem compartilhar dados sobre alcançabilidade e o estado dos seus enlaces (vizinhança). A principal vantagem do roteamento dinâmico é a capacidade de se adaptar a mudança de topologia sem depender da interferência humana. Todavia, o uso dos algoritmos de roteamento dinâmico insere tráfego de controle adicional na DCN. Além disso, período de convergência, eventuais *loops* podem ocorrer, enquanto os *switches* e roteadores estão processando suas atualizações nas tabelas. O período de convergência é dependente da topologia da rede e da velocidade de processamento dos equipamentos envolvidos, podendo variar em milésimos de segundos até vários minutos [Barreto 2008]. Pela inexistência de rotas, pacotes podem ser perdidos. Ademais, podem ocorrer incremento de latência, gerado por rotas mais longas entre os pares.

Os algoritmos de roteamento também podem ser classificados quanto ao seu método fundamental para transmissão de dados na rede, que podem ser: unicast, broadcast e multicast [Hosseini et al. 2007]. Os algoritmos que tratam da transmissão unicast tradicionalmente envolvem a comunicação *one-to-one*, ou seja, a troca de dados entre um par específico de *hosts* (uma origem e um único destinatário). Por sua vez, o tráfego de *broadcast* envolve a transmissão simultânea do mesmo pacote a partir de uma origem para todos os *hosts* de uma rede (*one-to-all*), portanto, este tipo de transmissão deve ser evitado em DCNs, principalmente por questões de segurança, ou seja, isolamento entre os inquilinos do DC. Já os algoritmos de multicast, basicamente, são destinados a transmissão do mesmo pacote de uma origem para um conjunto específico de *hosts* na rede (*one-to-many*). Todavia, podem também suportar a transmissão de várias origens para vários destinatários (*many-to-many*). Algoritmos de roteamento multicast são mais complexos, pois envolvem não só o conhecimento dos destinatários, mas também a cópia dos pacotes a partir da interface de entrada e o encaminhamento para todas as interfaces de saída. O que permite reduzir o tráfego nos enlaces melhorando a utilização da banda disponível na rede.

2.3.1 Técnicas para encaminhamento de pacotes

Operadores de DCN enfrentam desafios significativos no gerenciamento e configuração do DC [Kim et al. 2008]. Estes problemas advém do fato de que os protocolos de rede tradicionais não conseguem atender por completo os requisitos das DCNs. A dicotomia de alto nível é entre a criação de redes de Camada 2 ou Camada 3, cada um deles com vantagens e desvantagens associadas [Ramos 2013]. A seguir vamos apresentar resumidamente uma visão geral das principais técnicas de encaminhamento de pacotes.

O Ethernet se destaca como uma das tecnologias de rede mais utilizadas. Este protocolo tem características atraentes, especialmente para administradores de redes, devido sua simplicidade e facilidade de configuração [Kim and Rexford 2007]. Neste protocolo, os *hosts* finais já têm endereços globalmente únicos (endereços MAC de 48 bits), sem necessidade de nenhuma configuração. Além disso, os *switches* Ethernet aprendem automaticamente a localização dos *hosts*, reduzindo ainda mais a necessidade de configuração da rede. Outro ponto positivo desta tecnologia é que os endereços planos permanentes simplificam o suporte à mobilidade, à resolução de problemas de rede, e à aplicação de políticas de controle de acesso. Logo, a arquitetura Ethernet "pura" seria extremamente atraente, não fossem suas sérias limitações de escalabilidade [Ramos 2013].

O Ethernet depende de transmissão *broadcast* para suportar serviços essenciais como o ARP (*Address Resolution Protocol*) e o DHCP (*Dynamic Host Configuration Protocol*). Isso consome recursos em excesso, e também apresenta vulnerabilidades de segurança [Myers et al. 2004]. O Ethernet se baseia em inundação (*flooding*) para entregar os quadros a destinos desconhecidos. A sobrecarga de controle necessária para divulgar informações de cada máquina através de *flooding* pode ser muito grande, desperdiçando largura de banda e recursos de processamento [Kim et al. 2008]. Ademais, as tabelas de encaminhamento nos *switches* podem crescer muito, pois o plano de endereçamento aumenta o tamanho das tabelas proporcionalmente ao número total de *hosts* na rede.

Outra desvantagem da utilização do Ethernet é que, ele geralmente exige um mecanismo de proteção de circuito, como *Spanning Tree Protocol* (STP). O protocolo *Spanning Tree* automaticamente evita *loops*, impedindo que pacotes *broadcast* circularem continuamente na rede. Contudo, o STP leva a caminhos sub-utilizados e consequentemente distribuição irregular de cargas.

Uma forma de aumentar a escalabilidade do Ethernet é a construção de redes com várias LANs interligadas através de roteamento via *Internet Protocol* (IP). Nesta arquitetura híbrida, cada LAN contém no máximo algumas centenas de *hosts* que formam uma sub-rede IP. A cada sub-rede é atribuído um prefixo IP que a representa, e para cada *host* é atribuído um endereço IP com o prefixo da sub-rede. Ao contrário de um endereço MAC, que funciona como um identificador do *host*, um endereço IP denota sua localização na rede. Tal atribuição vai permitir tabelas de encaminhamento relativamente pequenas em todos os *switches* do DC [Ramos 2013]. O OSPF [Moy 1998] pode ser empregado para encontrar os caminhos de menor custos entre os *hosts*, na ocorrência de falhas, este pode compartilhar o estado dos enlaces, evitando o uso de caminhos indisponíveis.

Infelizmente, essa arquitetura híbrida também impõe suas limitações. O maior problema é a sua enorme sobrecarga de configuração. A atribuição de prefixos para as sub-redes, e a configuração das sub-redes nos roteadores normalmente são processos manuais, pois a atribuição deve seguir a hierarquia de endereçamento, e deve ser planejada de forma a reduzir o espaço de endereços desperdiçados, considerando também o uso futuro

de endereços para minimizar uma reconfiguração posterior [Ramos 2013]. Além do mais, o processo de adicionar um novo elemento de rede, geralmente, requer a configuração manual por parte do administrador de rede. O envolvimento de administradores humanos aumenta o tempo de reação à falhas e possíveis erros de configuração.

Da mesma forma, uma máquina virtual (VMs - *Virtual Machine*) ou *container* não pode acessar a rede até que seja configurado com um endereço IP correspondente à sub-rede onde está atualmente localizada (*host*). A utilização de DHCP automatiza a configuração, mas introduz uma sobrecarga considerável de gerenciamento. A falta de sincronia entre um servidor DHCP e um identificador de sub-rede de um roteador pode levar a *hosts* inacessíveis e erros difíceis de diagnosticar [Ramos 2013]. Por consequência, pode-se observar que o uso do roteamento IP leva a uma enorme sobrecarga de configuração e dificulta a alocação aleatória de VMs dentre diferentes servidores físicos [Kim et al. 2008]. Portanto, as abordagens tradicionais de redes não foram projetadas para atender as necessidades atuais de ambientes virtualizados, não sendo flexíveis o suficiente para suportar essas novas exigências.

Outro desafio é que, o advento da virtualização, da conteinerização e da Internet das Coisas (*Internet of Things - IoT*) tem gerado um crescimento explosivo no número de pontos finais na rede (*endpoints*). Para suportar este crescimento, comumente encontramos nas redes atuais o uso da agregação de endereços, como por exemplo, agregação de subredes IP. Todavia, a agregação descarta o controle refinado dos micro-fluxos com o objetivo de reduzir os estados de encaminhamento nas tabelas. Uma forma de manter os benefícios do controle de micro-fluxo é utilizar o encaminhamento baseado em rótulos ou *tags*, são exemplos: MPLS [Rosen et al. 2001], VLAN [IEEE 2009], VXLAN [Mahalingam et al. 2013]. Essas técnicas permitem manter o controle dos micro-fluxos sem um grande crescimento dos estados de encaminhamento e sem o uso de agregação [Hari et al. 2015]. Maiores detalhes sobre o roteamento baseados em rótulos e as técnicas de encaminhamento pertinentes a cada proposta serão discutidas no Capítulo 6.

2.4 Resiliência em Redes

Um DC é uma infraestrutura composta por um conjunto de equipamentos, *hardware* e *software* que como qualquer outro empreendimento está sujeito à falhas. No entanto, diferente de outras atividades, o *métier* do DC é prover uma infraestrutura **confiável** aos inquilinos. Falhas em grandes centros como Amazon EC2 geram prejuízos enormes, cerca de $ 5,600 por minuto [Gill et al. 2011]. Logo, a confiabilidade em uma Rede de *Data Center* tem como objetivo restabelecer a conectividade quando ocorrer uma falha na infraestrutura. Este processo requer mecanismos de recuperação rápidos e eficientes.

Nesta tese, definimos o termo resiliência como a habilidade de prover ou manter conectividade da rede na presença de falha em relação ao modo normal de operação. Em

[Gill et al. 2011] foi realizado um estudo no qual avaliou-se uma dezena de *Data Centers* durante o período de um ano em uma topologia convencional em camadas. Este estudo constatou a ocorrência de falha de pelo menos um enlace em um intervalo de 05 (cinco) minutos, seja entre os *switches* de borda, agregação ou núcleo. Outro importante resultado apontado neste estudo foi que 41.2% das falhas nos enlaces acontece em momentos em que os enlaces não estão trafegando nenhuma informação, ou seja, em quase metade das falhas não era necessário existir entradas adicionais nas tabelas para os caminhos alternativos. Todavia, mensurou-se que 28.6% das falhas tem impacto direto no tráfego da rede, tanto pelo tempo de convergência do protocolo de roteamento, quanto pelo redirecionamento dos pacotes por caminhos alternativos. Portanto, novos modelos de roteamento resiliente se fazem necessários.

Para manter a conectividade nas DCNs precisamos utilizar mecanismos de recuperação. A seguir descreveremos os princípios de detecção e os principais mecanismos tipicamente encontrados na literatura que podem ser aplicados.

2.4.1 Ciclos de recuperação e reversão de falhas

O princípio básico de um sistema de recuperação é simples. Em condições normais (isto é, sem falhas), o tráfego é transportado ao longo do caminho primário. Se for detectada uma falha ao longo desse caminho, o esquema de recuperação é ativado. Uma parte do percurso (ou todo ele, dependendo da técnica de recuperação), será contornado por um caminho alternativo. Na maioria dos casos, é usado um caminho de recuperação disjunto do caminho principal, para assegurar que uma falha não afete ambos os caminhos [Ramos 2013].

Em [Vasseur et al. 2004, Jorge and Gomes 2006, Kvalbein et al. 2009], encontram-se uma enorme variedade de esquemas de recuperação. Todos eles apresentam um fluxo semelhante de fases, denominado ciclo de recuperação, conforme ilustrado na Figura 2.3.

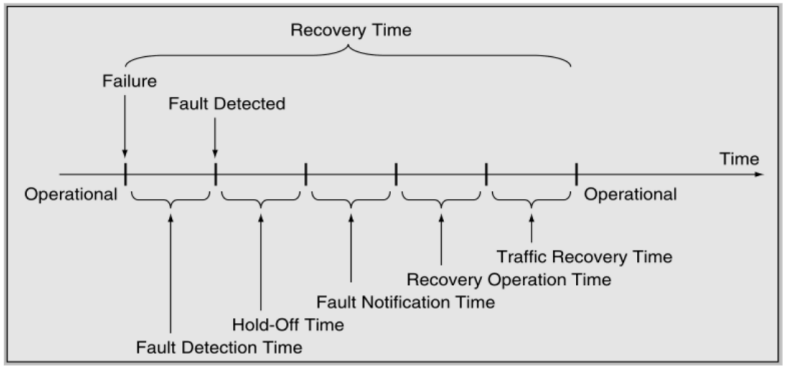


Figura 2.3: Ciclo de recuperação. Fonte: [Vasseur et al. 2004].

A Figura 2.3 ilustra as fases do ciclo de recuperação. Resumidamente, caso aconteça uma falha na rede, algum tempo poderá ser demandado até que um elemento próximo a detecte. Este tempo pode variar conforme: frequência de sinal transmitido, velocidade

de detecção da falha e notificação, além do tempo que o elemento leva para coletar e processar todas essas informações [1].

O elemento que a detectou pode (ou não) esperar algum tempo antes de começar a enviar mensagens de notificação para os outros elementos da rede. Este tempo de espera poderia permitir que um esquema de recuperação de uma camada inferior reparasse a falha. Se a falha persistir após o tempo de espera, mensagens de notificação são enviadas em toda a rede para informar aos outros elementos que estarão envolvidos na ação de recuperação [Ramos 2013].

É importante observar que o tempo de espera pode ser de natureza estática. Portanto, um intervalo fixo será utilizado para todos os casos. Um método dinâmico também pode ser utilizado para definir o tempo de espera. Neste último, um temporizador pode ser usado em função do número de falhas dentro de um determinado período. Quanto mais falhas forem detectadas, maior será o tempo de espera. Esta método é denominado de amortecimento (*dampening*), pois ajuda a estabilizar a rede no caso de um recurso instável (isto é, um recurso que alterne rapidamente entre o estado de funcionamento e estado de falha) [Vasseur et al. 2004].

Conforme ilustrado na Figura 2.3, o tempo entre a primeira e a última ação de recuperação é chamado o tempo de operação de recuperação. Este intervalo de tempo pode incluir a troca de mensagens entre os diferentes elementos envolvidos na ação de recuperação para coordenar a operação. Após a última ação de recuperação o tráfego começa a usar o caminho de recuperação. No entanto, ele ainda pode levar algum tempo antes que o tráfego esteja completamente recuperado. Este tempo de recuperação do tráfego pode depender do atraso de propagação ao longo do caminho de recuperação, a localização da falha, e o esquema de recuperação usado [Ramos 2013].

Após o ciclo de recuperação, a rede estará operacional. Todavia, as novas rotas de tráfego ao longo dos caminhos de recuperação podem ser piores, ou seja, caminhos de recuperação mais longos ou mais congestionados que o caminho original.

Para otimizar o uso dos recursos pós-recuperação, protocolos dinâmicos de reencaminhamento podem ser usados. Outra possibilidade é esperar a reparação do defeito que tenha ocorrido e redirecionar o tráfego do caminho de recuperação de volta para o percurso original, uma vez que a falha esteja completamente reparada. Esta operação também segue uma sucessão de fases, denominada ciclo de reversão [Vasseur et al. 2004].

As diferentes fases deste ciclo são apresentadas na Figura 2.4. O ciclo de reversão têm uma forte semelhança com o ciclo de recuperação, descrito anteriormente. Uma vez que a falha esteja reparada, pode haver alguma demora, por exemplo, dependendo de protocolos de camada inferior, antes do reparo ser detectado. Depois disso, o protocolo pode decidir esperar durante um certo tempo antes de iniciar a notificação de reparação

[1] Tipicamente, a detecção da falha acontece em camadas inferiores da rede, que em seguida, notifica as camadas superiores de rede [Vasseur et al. 2004].

do defeito. Este tempo pode ser necessário para assegurar que o caminho é estável, pois no caso de uma falha intermitente, uma reação rápida de processo de reversão pode conduzir a condições de rede instáveis. Tal como no caso anterior, note que o temporizador pode ser estático ou dinâmico (amortecimento) [Ramos 2013].

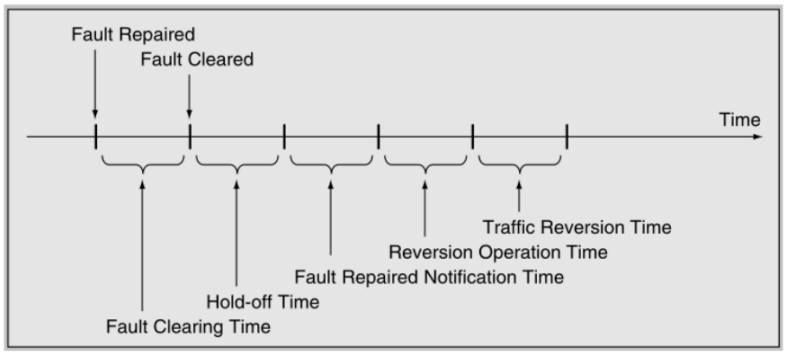


Figura 2.4: Ciclo de reversão. Fonte: [Vasseur et al. 2004].

Depois disso, de maneira semelhante ao ciclo de recuperação, a falha reparada é notificada em toda a rede e a operação de reversão é executada, o tráfego é comutado novamente a partir do caminho de recuperação para o percurso original. Finalmente, pode levar algum tempo antes que o tráfego comece a fluir normalmente no caminho original (tempo de reversão de tráfego) [Vasseur et al. 2004].

Observe que em contraste com o ciclo de recuperação, que é tipicamente a reação a um acontecimento inesperado, a falha do ciclo de reversão pode ser planejada com antecedência. Em uma reversão como não há necessidade de uma operação precipitada, um mecanismo bem controlado com o mínimo de interrupção será mais eficiente.

2.4.2 Estratégia dos mecanismos de recuperação e suas características

Mecanismos de recuperação podem ser categorizados em uma enorme variedade de diferentes abordagens. Cada uma delas apresenta vantagens e desvantagens. A seguir descrevemos os principais pontos que devem ser observados nos mecanismos de recuperação:

Rotas Proativas versus Reativas: de forma geral, esta seleção está associada ao tipo de recuperação da rota que será escolhida. Na opção de rota proativas (pré-calculada), para todas as situações de falha ocorridas na rota do fluxo, a recuperação é calculada antecipadamente, ou seja, antes de qualquer falha ocorrer. Já na opção reativa (dinâmica) a rota de recuperação não é planejada, ou seja, sua definição se dará uma vez que a falha é detectada. Se ocorrer uma falha, o mecanismo de recuperação começa a procurar de forma reativa por possíveis rotas alternativas em toda a rede.

A vantagem da opção de rotas proativas é que estas permitem uma rápida recuperação da falha, enquanto que um mecanismo de recuperação reativo pode demorar mais tempo para identificar uma rota de recuperação adequada. Por outro lado, a opção por uma rota proativa carece de flexibilidade para cenários não planejados. Um mecanismo reativo é capaz de procurar uma rota de recuperação mesmo para cenários não contabilizados, embora não haja nenhuma garantia de que ele vai encontrar essa rota de recuperação [Ramos 2013].

Proteção versus Restauração: uma distinção muito importante a ser feita ao considerar mecanismos de recuperação é entre proteção e restauração. Ambas as opções exigem sinalização, mas a sutil diferença está no momento das ações de sinalização. No caso da proteção, os caminhos de recuperação são pré-calculados e totalmente sinalizados antes de ocorrer uma falha. Assim, quando a falha ocorre, nenhuma sinalização adicional é necessária para estabelecer o caminho de proteção. No caso da restauração, os caminhos de recuperação podem ser pré-calculados ou dinâmicos, mas quando ocorre uma falha, será necessária sinalização adicional para estabelecer o caminho de restauração [Ramos 2013].

Uma grande vantagem da proteção em comparação com a restauração é tipicamente o tempo de recuperação rápida. A sinalização adicional, após uma ocorrência de falha no caso de restauração pode consumir um tempo precioso. Por outro lado, técnicas de restauração podem ser mais flexíveis em relação aos cenários de falha que podem ser recuperados.

Local versus Global: para contornar os pontos de falha, esquemas de recuperação mudam a rota do tráfego afetado. Na recuperação local, apenas os elementos de rede afetados são contornados, alterando o mínimo possível a rota original do fluxo. Se um único enlace falhar, um caminho de recuperação se estabelece entre os nós adjacentes à falha. No outro extremo temos a recuperação global, onde o caminho completo entre a origem e o destino é trocado pelo caminho de recuperação [Ramos 2013]. Naturalmente que essas duas abordagens representam apenas os dois extremos de um grande número de possibilidades intermediárias, onde a extensão da recuperação é maior que na abordagem local, porém mais curto na global. Em [Sharma and Hellstrand 2003] na rede G-MPLS as opções intermédias são chamadas de recuperação do segmento.

A recuperação local é, geralmente, mais rápida que a global, sendo essa uma vantagem importante em aplicações sensíveis à latência. Por outro lado, na recuperação local a rota resultante após a recuperação pode ser mais longa do que o caminho original. Além disso, se dois enlaces sucessivos no caminho principal falharem, a recuperação global ainda resolverá o problema, já a recuperação local, não [Ramos 2013].

Centralizado versus Distribuído: os mecanismos de recuperação centralizados dependem de um controlador central para determinar quais ações de recuperação devem ser tomadas, a partir de uma visão global do estado da rede. Este controlador determina onde e quando uma falha ocorreu, reúne informações de estado de toda a rede e comanda

a reconfiguração de todos os elementos envolvidos no processo de recuperação. Sistemas de gerenciamento baseados em Redes Definidas por Software geralmente empregam mecanismos centralizados.

Já os mecanismos de recuperação distribuídos operam sem a intervenção de um sistema de controle central. Neste caso, os elementos de rede possuem sistemas de controle inteligentes, que iniciam de forma autônoma as ações de recuperação. Assim, o controle é distribuído ao longo dos elementos de rede envolvidos no processo de recuperação. Em contraste com os mecanismos centralizados, estes sistemas de controle distribuído não têm uma visão global, mas apenas uma visualização local do estado da rede. Eles podem ter de trocar mensagens entre si para fornecer informações suficientes e coordenar as ações de recuperação. Assim, múltiplos elementos de rede trabalham em paralelo para alocar o tráfego interrompido em uma rota alternativa [Sharma and Hellstrand 2003, Ramos 2013].

2.5 Redes Definidas por Software

Um conceito promissor de redes programáveis referido como Redes Definidas por Software (SDN - *Software Defined Networks*) tem-se fortalecido. Esse conceito de SDN apoia-se na separação do plano de controle do plano de dados, permitindo que um controlador central interaja diretamente com os ativos da rede. Recursos de controle, monitoramento de tráfego, algoritmos de roteamento e chaveamento de pacotes são exemplos de funcionalidades de *software* que podem ser implantadas na rede, através do controlador [Kotronis et al. 2012].

Para tornar possível a implementação desse novo conceito, surgiu uma interface, chamada OpenFlow (OF) [McKeown et al. 2008], que é um padrão aberto que permite a programação dos elementos ativos da rede, tais como roteadores, *switches* ou pontos de acesso sem fio. O OpenFlow é uma proposta pragmática com grande potencial para suportar o grau de programabilidade que a SDN necessita.

Lantz [Lantz et al. 2010] explica que em uma SDN, o plano de controle (ou "sistema operacional de rede") é separado do plano de dados. Normalmente, o sistema operacional de rede observa e controla o estado de toda a rede a partir de um ponto centralizado, oferecendo recursos como: protocolos de roteamento, controle de acesso, virtualização de rede, gestão de energia e também protótipos de novos de protocolos. A principal consequência da SDN é que as funcionalidades da rede podem ser alteradas ou mesmo definidas após a rede ter sido implantada. Novas funcionalidades podem ser adicionadas, sem a necessidade de modificar o *hardware*, permitindo que o comportamento da rede evolua na mesma velocidade que o *software*.

Casado [Casado et al. 2012] apresenta características desejáveis para SDN, separando-as em características de *hardware* e *software*. O *hardware* deve ser simples, barato, fácil de operar e independente de fabricante, evitando que os usuários sejam obrigados a usar

equipamentos de um único fornecedor. Também deve suportar inovações futuras, evitando atualizações desnecessárias. Com relação ao *software*, Casado expõe que o mesmo deve ser flexível, estruturado e suportar uma ampla variedade de requisitos (isolação, virtualização, engenharia de tráfego, controle de acesso, etc). Também deve ser modular e expansível, permitindo inclusão e alteração de módulos.

Um dos pontos fortes das SDN é a visão centralizada da rede, sobre a qual é possível desenvolver análises detalhadas, tomando-se decisões operacionais sobre como o sistema deve operar. Observa-se que essa noção de visão centralizada da rede é uma visão lógica, não necessitando que o controlador esteja fisicamente localizado em um ponto único do sistema. A abstração de visão global pode ser implementada de forma distribuída, seja pela divisão dos elementos da visão entre domínios diferentes, seja pela implementação de um controlador distribuído [Guedes et al. 2012, Spalla et al. 2016].

Os controladores permitem a visualização dos eventos gerados pelas interfaces, sendo responsável também por adicionar e remover as entradas das tabelas de fluxo nos equipamentos utilizados na SDN. Pode-se citar como exemplo um controlador simples, com fluxos estáticos interligando um grupo conhecido de computadores. Nesse caso teríamos uma SDN semelhante a uma rede utilizando a tecnologia de VLANs [McKeown et al. 2008]. É possível também a existência de controladores mais sofisticados que adicionam e removem fluxos dinamicamente, de acordo com a necessidade da rede. Controladores podem suportar múltiplos pesquisadores, com diferentes contas e permissões, executando vários experimentos independentes, todos em uma mesma rede física.

Outra inovação apresentada pela SDN é a inclusão de uma camada de abstração de *hardware*, similar ao que acontece na virtualização de computadores. Esta camada de virtualização permite que a rede seja divida em fatias completamente distintas, executadas simultaneamente sem qualquer interferência umas nas outras. A camada de virtualização permite que, acima dela, sejam adicionados novos protocolos e formatos de endereçamento. Abaixo, novos modelos de *hardware* podem ser desenvolvidos, otimizados para diferentes ambientes, velocidades ou tipos de mídia (com fio e sem fio) [Guedes et al. 2012]. Percebe-se que essa camada de virtualização possibilita às SDN alcançar as características de *software* indicadas por Casado.

Por sua vez, os *switches* virtuais, são aplicações construídas com a finalidade de encaminhar pacotes utilizando a pilha de rede do sistema operacional do *host* onde são executados, capazes de transformar um computador, ou uma porção dele (ex. uma máquina virtual) em um encaminhador de pacotes. Permitem anexam uma interface interna presente na aplicação à uma interface de rede física (Ethernet) ou lógica (ex. *veth*, *tap*, entre outras) do sistema, criando assim uma ponte entre conexão interna/externa à aplicação. Com isso, os dados que seriam direcionados à uma porta física ou lógica são encaminhados para essa interface interna, permitindo assim o tratamento desses dados de acordo com as premissas do *switch* virtual, que pode variar desde ao simples encami-

nhamento de pacotes de rede como em uma *bridge*, à classificação dos pacotes através da análise de cabeçalhos dos protocolos de rede, elevando o grau de abstração no tratamento e encaminhamento de pacotes de dados na rede.

2.6 Considerações Finais

O objetivo deste capítulo foi descrever os conceitos básicos para o entendimento da arquitetura RDNA. Na Seção 2.1 descrevemos resumidamente uma visão geral da Teoria dos Números, algoritmo de Euclides, Algoritmo de Euclides Estendido, fatoração única, aritmética modular e sistema de congruência, conceitos esses necessários para o entendimento do Teorema Chinês do Resto, que é o alicerce da nossa arquitetura.

Em seguida na Seção 2.2 apresentamos uma síntese sobre Redes de *Data Center*, focando nas abordagens centradas em comutadores e servidores. Na Seção 2.3 abordamos as principais características dos protocolos e dos algoritmos de roteamento, além das técnicas de encaminhamento de pacotes comumente utilizadas.

Uma das principais contribuições da nossa arquitetura esta em prover proteção fim-a-fim com reação rápida à falha sem a necessidade de troca de mensagens de controle no núcleo da DCN. Portanto, na Seção 2.4 explicamos os ciclos de recuperação e reversão de falha, além de descrever algumas das principais estratégias dos mecanismos de recuperação disponíveis na literatura. Por último, na Seção 2.5, descrevemos de forma resumida uma visão geral das Redes Definidas por *Software* e seus principais componentes.

Capítulo 3

RDNA: Arquitetura Definida por Resíduos para Redes de *Data Centers*

Este capítulo apresenta os fundamentos da nova arquitetura, denominada RDNA: Arquitetura Definida por Resíduos para Redes de *Data Centers*, projetada para atender as demandas dos serviços de conectividade para Computação em Nuvem. A RDNA desfruta de características habilitadoras rumo a Redes de *Data Centers* confiáveis, oferecendo um mecanismo original de proteção completa da rota sem utilizar tabelas no núcleo da topologia. A topologia é estruturada em dois níveis (2-tier), viabilizando a escalabilidade do nosso modelo de encaminhamento baseado em resíduo.

3.1 Visão geral

Em nossa arquitetura a rota é definida a partir de um conjunto de resíduos, calculado através de um sistema de congruência linear usando algoritmos tradicionais como: algoritmo Euclidiano Estendido e o Teorema Chinês do Resto. Graças a essas propriedades matemáticas, nosso modelo de encaminhamento unicast funciona baseado em uma simples operação de resto de divisão, entre os identificadores codificados dentro dos pacotes e um único identificador armazenado no próprio *switch* de núcleo. Da mesma forma, o encaminhamento multicast utiliza as mesmas operações, resultando em valores de coeficientes para uma função polinomial específica, que permite construir árvores multicast de forma escalável, sem nenhuma tabela nos *switches* de núcleo e sem reescrita do cabeçalho do pacote a cada salto.

A RDNA avança no estado da arte propondo um *mecanismo de reação rápido à falha*. Este mecanismo permite encaminhar o pacote por uma rota disjunta específica, sem a necessidade de aguardar mensagens de controle/sinalização, a exemplo do que acontece com os protocolos habituais, como: STP, OSPF e MPLS Fast Reroute, eliminando assim a latência de convergência na recuperação da rede. Isso só é possível em razão do

nosso modelo de encaminhamento, projetado sobre dois identificadores de rota, codificados no pacote, portanto, cada *switch* é capaz de redirecionar o tráfego de forma proativa e independente, sem incrementar a latência da comutação dentro do núcleo da topologia.

Uma entidade de gerenciamento (controlador de rede) é responsável por encontrar as melhores rotas para comunicação unicast e multicast. Além disso, esta entidade tem a capacidade de fornecer enorme flexibilidade na movimentação das máquinas virtuais e *containers* que executam os serviços e as aplicações dos inquilinos. Este recurso, muito desejado na orquestração em DCNs, é obtido em consequência da separação semântica dos endereços de identificação e localização. Basicamente, em nossa arquitetura consideramos separar o localizador do identificador de modo que seja transparente para as VMs e compatível com o *hardware* existente. Nosso modelo de encaminhamento não usa endereços IP para localizar as VMs dentro da *Data Center*. O endereço físico MAC e IP representam a identificação da VM. Desta forma, ao contrário da atribuição hierárquica tradicional de endereços IP, não há restrições sobre como os endereços serão alocados. Por sua vez, o localizador da VM é representado pelo identificador de rota, que é calculado na entidade de gerenciamento e inserido dentro do pacote.

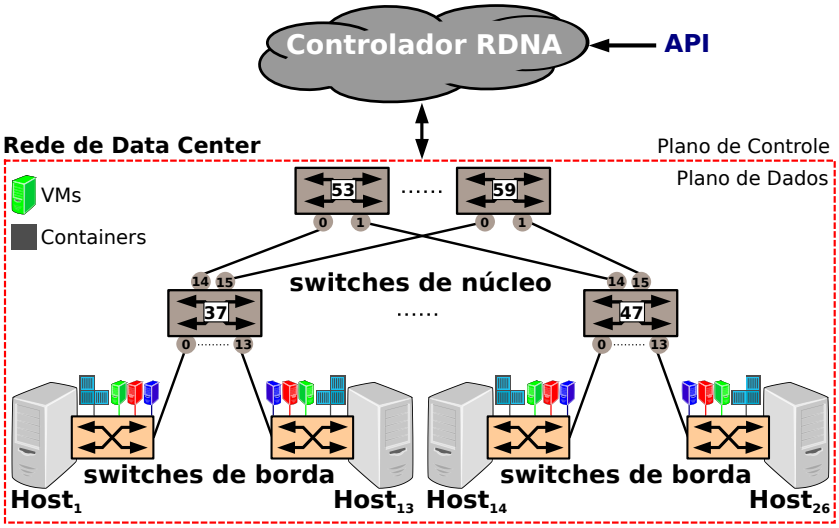


Figura 3.1: Projeto da arquitetura RDNA.

Um ponto chave em nossa arquitetura foi considerar o desacoplamento entre os *switches* de borda e núcleo, que foram posicionados em sintonia com a estrutura topológica utilizada. Esta escolha permitiu definir um plano de dados composto por dois blocos fundamentais autônomos, concretizando um padrão real de arquitetura SDN pragmática apropriada para Redes de *Data Centers*. Os *switches* de borda armazenam os estados e adicionam identificadores de rotas dentro dos pacotes, deixando os *switches* de núcleo livres para comutar pacotes de maneira simples e eficaz.

A Figura 3.1 sumariza a arquitetura RDNA, ilustrando os elementos funcionais, sendo: i) controlador logicamente centralizado, que define as políticas de rede; ii) *switches* de

borda, que codificam as informações de rota dentro dos pacotes; e iii) *switches* de núcleo sem tabela, que encaminham os pacotes baseados em operações matemáticas clássicas. A seguir descrevemos resumidamente as principais atribuições dos elementos na arquitetura RDNA.

- **Controlador RDNA:** é uma entidade central ciente da topologia completa da rede, que é descoberta usando, por exemplo, um protocolo de descoberta de camada de enlace (*Link-Layer Discovery Protocol* - LLDP). Este elemento atua soberano no plano de controle e é responsável por: i) calcular e enviar um identificador único para cada *switch* de núcleo que é utilizado em todo processo de encaminhamento unicast e multicast; ii) calcular e enviar o grau da expressão polinomial conforme arranjo topológico; iii) definir as funções e políticas de rede (por exemplo, proteção de rota) para fluxos específicos; iv) definir as rotas de rede para comunicação unicast e multicast utilizando algum algoritmo de roteamento que encontre o melhor caminho; v) calcular o identificar de rota (R_i) entre cada par de *hosts* (ou mesmo máquinas virtuais e *containers*), além de calcular todas as árvores multicast (X_i); vi) enviar os R_i e X_i para os *switches* de borda com os estados de cada fluxo, de maneira que possa representar um completo mapeamento para comunicação unicast e multicast, respectivamente; e além disso, vii) deve fornecer uma interface de programação (API) para suportar a comunicação com solicitações externas.

- ***Switch* de núcleo:** o propósito principal deste elemento é fornecer um comutador eficiente de pacotes para comunicação unicast e multicast em uma Rede de *Data Center*. Portanto, este elemento deve ser robusto, garantindo latência de encaminhamento determinística mesmo em momentos de instabilidade da rede, ademais, deve operar de maneira simples, inteligente e sustentável. Para atender este propósito, o primeiro passo é substituir as tradicionais operações de consulta baseada em tabela (Camada 2 e 3), por um mecanismo de encaminhamento sem tabela. Esta substituição elimina a necessidade de utilizar TCAM, que é um componente especializado de alto custo e alto consumo de energia elétrica. Isto posto, na comunicação unicast nosso modelo de encaminhamento no *switch* de núcleo opera utilizando apenas *operações de resíduos*, isto é, o resto da divisão, entre os identificadores de rota, incorporadas dentro dos pacotes, denotado por (R_i), e um identificador único pertencente ao *switch*, denotado por (S_i). Esses identificadores de *switch* (S_i), não são números arbitrários. Eles precisam ser: i) coprimos entre si, ou seja, tais números devem possuir como divisor comum apenas a unidade 1; e ii) ele deve ser maior que a quantidade de interfaces físicas no respectivo *switch*. Para a comunicação multicast o encaminhamento no *switch* de núcleo opera basicamente da mesma maneira, a única diferença é que o resto da divisão entre os identificadores da árvore multicast, denotado por (X_i) e o S_i indicam valores para os coeficiente de uma expressão

polinomial específica, cujo o resultado representa um conjunto de portas de saída, que receberão cópias do respectivo pacote a ser encaminhado.

- ***Switch* de borda:** é o elemento com o papel de armazenar estados de fluxo e mapear os identificadores de rota R_i e X_i nos pacotes. Posicionados estrategicamente dentro do *host* físico, este elemento é facilmente suportado em *software*, não existindo significativa restrição no tamanho e na quantidade de tabelas de fluxo. Basicamente, o mapeamento significa que um determinado R_i ou X_i deve ser incorporado nos pacotes pelo elemento de borda. Atualmente encontramos diversos *switches* em *software* ou *vSwitches* que fornecem este recurso, como por exemplo: CPqD Softswitch13 [CPqD 2014], Click Modular Router [Kohler et al. 2000] e Open vSwitch [Pfaff et al. 2015]. O *vSwitch* ocupa uma posição única privilégiada na pilha de rede, pois pode facilmente modificar pacotes sem exigir alterações nas VMs e *containers* dos inquilinos ou nas camadas de transporte. A funcionalidade incorporada no *vSwitch* pode ser informada dos recursos subjacentes do *hardware* disponível, por exemplo, apresentados pela interface de rede (NIC) e pelo Sistema Operacional do *host*. No *switch* de borda, esses estados podem ser configurados para mapear os identificadores utilizando campos existentes do pacote (estados por fluxo) sem gerar nenhuma sobrecarga significativa ou prejuízo com perda de espaço útil de transferência dos dados pela rede. Uma abordagem reativa ou proativa pode ser usada para armazenar os estados no *switch* de borda. No primeiro caso, as regras são instaladas quando o *switch* de borda de ingresso encaminha o pacote para o controlador RDNA via algum protocolo de comunicação e solicita o mapeamento da rota para a comunicação unicast (R_i) ou multicast (X_i). No segundo caso, o controlador RDNA pré-calcula a rota e entrega antecipadamente o estado de entrada por fluxo, definindo as rotas entre as VMs e *containers* dentro do *Data Center*. Em nossa arquitetura, também é função deste elemento conectar uma rede de domínio RDNA com outro domínio não RDNA, ou seja, o *switch* de borda possui a funcionalidade de um *gateway* de conexão com outros domínios de forma transparente e flexível. Desta maneira, quando os pacotes contendo os dados dos dispositivos IoT chegarem até o DC, por exemplo, cabe ao *switch* de borda inserir as rotas resilientes para a comunicação unicast ou a rota codificada da árvore multicast. Esta ação em conjunto com nosso mecanismo inovador de reação à falha, garantirá encaminhamento determinístico na comutação *host* a *host*, permitindo por exemplo, que serviços de missão crítica sejam minimamente impactados em período temporário de falha. Para a comunicação multicast a codificação da árvore beneficiará os serviços de grupo, minimizando o uso dos enlaces de forma eficiente.

Nas próximas seções detalharemos o funcionamento do nosso sistema de roteamento, além da codificação do identificador do *switch* de núcleo, de todos os identificadores de

rota para comunicação unicast e multicast. Ademais, também explicaremos o funcionamento dos mecanismos de encaminhamento sem tabela capazes de suportar o roteamento resiliente com latência de encaminhamento determinística para Redes de *Data Center*.

3.2 Sistema de Roteamento

Com o objetivo de simplificar o sistema de roteamento, e ao mesmo tempo garantir flexibilidade nas políticas de rede do *Data Center*, na RDNA optou-se em utilizar o roteamento na origem. Nossa estrutura permite que os *switches* de borda codifiquem identificadores de rota diretamente no cabeçalho dos pacotes sem significativa sobrecarga. Os pacotes são encaminhados salto a salto até o destino em uma rede de núcleo sem tabela. Além disso, garantimos uma rota de proteção completa também na origem, o que garante resiliência ao sistema de roteamento de maneira genuína e eficiente.

O modelo de encaminhamento baseado em resíduos fornece para nossa arquitetura a agilidade desejada rumo a um plano de dados eficiente. Apoiada pelo roteamento na origem, podemos facilmente aplicar qualquer política ou algoritmo de roteamento desejado nas Redes de *Data Center*, por exemplo, podemos escolher um algoritmo de roteamento que considere como melhor rota a menor distância (menor número de saltos) ou a soma mínima dos peso das arestas, para isso, basta utilizar algoritmos clássicos como: Dijkstra, Bellman-Ford [Bellman 1958] e Floyd-Warshall [Cormen et al. 1990] para definir o caminho. Qualquer um destes algoritmos pode ser facilmente aplicado em nosso plano de controle.

Nós formalmente definimos o domínio de uma rede RDNA consistindo no conjunto S de n *switches* em um caminho desejado, de modo que $S = S_i | i = 1, 2, 3, \cdots, n$. Seja p um conjunto de portas de saída $p = \{p_1, p_2, \cdots, p_k\}$, onde p_i são inteiros arbitrários representando o número da porta física (isto é, porta de saída para o pacote) no *switch* de núcleo S_i.

Para tornar o sistema de congruência solucionável, n inteiros $S_1, S_2, S_3, \cdots, S_n$ precisam ser coprimos. Portanto, existe um inteiro único R tal que $0 \leq R < \prod_{i=1}^{n} S_i$ que resolve a seguinte congruência apresentada na Equação 3.1:

$$\begin{aligned} < R >_{S_1} &\equiv p_1 \\ < R >_{S_2} &\equiv p_2 \\ &\vdots \\ < R >_{S_n} &\equiv p_k \end{aligned} \tag{3.1}$$

Podemos reescrever a Equação 3.1 de forma simplificada para $< a >_b \triangleq a \ modulo \ b$. Por exemplo, $a = 12$, $b = 5$, temos $< 12 >_5 = 2$, portanto, 2 é o restante da divisão entre 12 e 5.

Assim, o sistema de roteamento deve descobrir um valor de R (que explicite o identificador da rota), dado o conjunto de módulos S (identificador do *switch* de núcleo), e sua representação pelo RNS p (número das portas de saída).

Como o Teorema Chinês do Resto [Ding et al. 1996] (TCR) afirma que é possível reconstruir R através de seus resíduos em um RNS [Garner 1959], segue:

$$R =< \sum_{i \in S} p_i \cdot M_i \cdot L_i >_M, \tag{3.2}$$

onde:

$$p_i = R \; modulo \; S_i \tag{3.3}$$
$$M_i = \frac{M}{S_i} \tag{3.4}$$
$$L_i = < M_i^{-1} >_{v_i} \tag{3.5}$$

O número em p_i na Equação 3.3 representa o valor da porta de saída do pacote no *switch* S_i na comunicação unicast. Melhor dizendo, o resto da divisão entre o identificador de rota (R) e (S_i) deve indicar a porta de saída do pacote p_i. Na comunicação multicast o valor de p_i indica o valor de coeficiente da função polinomial, conforme detalharemos na Seção 3.7.

Na Equação 3.4 M_i recebe a divisão entre M, que é o produtório de todos os S_i que compõem a rota, ou seja,

$$M = \prod_{i \in S} S_i, \tag{3.6}$$

dividido pelo respectivo S_i. Finalmente, a Equação 3.5 significa que L_i é o inverso multiplicativo modular de M_i. Em outras palavras, L_i é um número inteiro tal que:

$$< L_i \cdot M_i >_{S_i} = 1 \tag{3.7}$$

Os conceitos aqui apresentados formam a base do nosso sistema de roteamento, nas próximas seções explicaremos o funcionamento do mecanismo de encaminhamento e como o controlador RDNA calcula os identificadores utilizados nos *switches* de núcleo, assim como os identificadores de rota para comunicação unicast e multicast.

3.3 Cálculo do Identificador do *Switch* de Núcleo

O conjunto S_i, dado n, pode ser definido utilizando a função Totiente de Euler, descrita em [Abramowitz 1974]. Este conjunto é composto por números inteiros positivos que em

nossa arquitetura representam os identificadores únicos utilizados nos *switches* de núcleo.

Conforme apresentado na Equação 3.8, onde $\phi(n)$ é um conjunto de inteiros positivos que não excedem n e que não têm divisores comuns com n (diferente do divisor comum 1). Em outras palavras, $\phi(n)$ é um número inteiro coprimo com n, onde o produto está sobre os números primos distintos dividindo n, tal que $1 \leq n \leq n$.

$$\phi(n) = n \prod_{P \text{ primo}} \prod_{P|n} \left(1 - \frac{1}{P}\right) \tag{3.8}$$

O cálculo dos identificadores para os *switches* de núcleo é realizado no controlador RDNA, que têm conhecimento de toda a topologia de rede. Além da Equação 3.8 o controlador também considera a restrição da quantidade de interfaces físicas em cada *switch*, ou seja, o identificador a ser atribuído ao respectivo *switch* deve ser maior ou igual ao número de portas existentes. Isso se faz necessário para garantir que o resto da divisão entre o identificador da rota (R_i) e o identificador do *switch* (S_i) resulte em um número que possa representar diretamente o número da porta (p_i) de saída do pacote, conforme apresentaremos a seguir em nosso sistema de roteamento.

3.4 Codificação do Identificador de Rota para Comunicação Unicast

Para ilustrar o conceito de comunicação unicast, considere o cenário mostrado na Figura 3.2, em que uma VM "verde", localizada no $host_1$ deseja se comunicar com outra VM "verde", localizada no $host_{56}$. Vamos utilizar em todos os exemplos deste capítulo a comunicação entre duas VMs dentro do próprio DC apenas para facilitar o entendimento da arquitetura. Usando um algoritmo de roteamento, o controlador RDNA (não conectado aos *switches* por questão de clareza) seleciona um caminho de ponta a ponta na rede de acordo com as políticas de rede do DC, conforme apresentado na Figura 3.2 (**Passo I**). Por exemplo, ele seleciona a rota definida através dos *switches* $S = \{37, 53, 47\}$ compondo o que chamamos de rota principal. Nesse caso, as portas de saída dos *switches* são $p = \{14, 3, 13\}$. Em seguida, calcula uma identificação de rota principal (IRP), logo, $IRP = 86446$. O controlador RDNA então, utilizando algum protocolo que habilita a comunicação entre o plano de controle e o plano de dados, envia a identificação de rota para os *switches* de borda que, em seguida, instalam as respectivas regras em sua tabela de fluxo. Posteriormente, o *switch* de borda de ingresso é responsável pela incorporação do Identificador de Rota Principal (IRP) em cada pacote (por exemplo, em um de seus campos do cabeçalho), proveniente da VM do $host_1$ *src* para a VM do $host_{56}$ *dst* (**Passo II**).

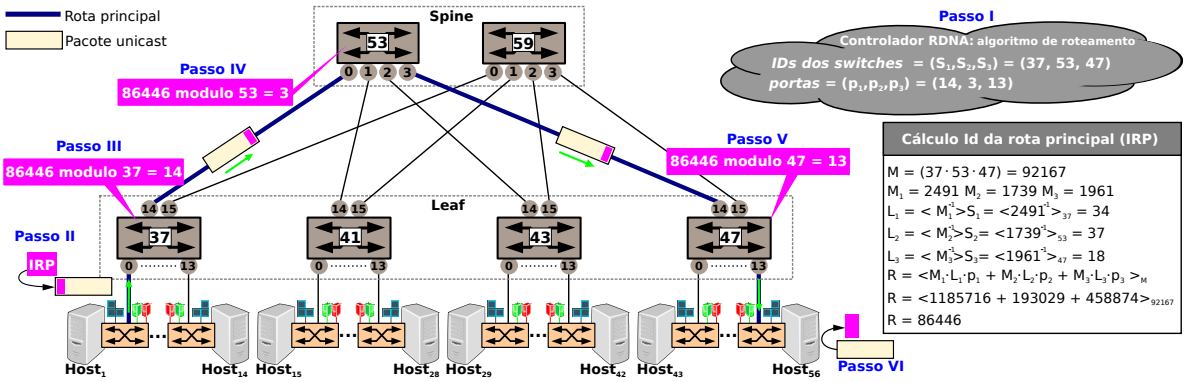


Figura 3.2: Sistema de roteamento RDNA para comunicação unicast.

Uma vez que o pacote tenha entrado no núcleo, a cada *switch* que o pacote chega, o resto da divisão entre a e b, denotada como $< a >_b$, entre o IRP codificado no pacote ($R = 86446$) e o respectivo *ID do switch*, é calculado para definir a porta de saída apropriada para encaminhá-lo. Assim, como ilustrado na Figura 3.2, quando S_{37} recebe um pacote com ID de rota ($R = 86446$), ele encaminha o pacote para a porta $< 86446 >_{37} = 14$ (**Passo III**); então, S_{53} encaminha para a porta $< 86446 >_{53} = 3$ (**Passo IV**); depois, S_{47} encaminha o pacote para a porta $< 86446 >_{47} = 13$ (**Passo V**), atingindo o *switch* de borda de egresso que remove o ID da rota do pacote (**Passo VI**) entregando em seguida para a VM "verde", localizada no $host_{56}$ *dst* (como pode ser visto na Figura 3.2).

Note que nenhuma da VMs envolvidas na comunicação precisou ser modificada ou mesmo ser notificada da alteração ocorrida dentro do cabeçalho do pacote. Cabe ao *switch* de borda deixar transparente qualquer modificação no pacote, sem prejudicar significativamente a transferência de dados entre as VMs. Do mesmo modo nenhum estado ou protocolo de convergência é requerido pelos *switches* de núcleo para realizar o encaminhamento dos pacotes, graças ao modelo de encaminhamento utilizado na arquitetura RDNA. Todavia, eventuais falhas temporárias podem ocorrer e os pacotes não possuem nenhuma informação que permita contornar esta falha. Por isso, na próxima seção, apresentaremos nossa abordagem que permite a rápida reação a falha em redes sem estado.

3.5 Codificação do Identificador de Rota para Comunicação Unicast Resiliente

Redes de *Data Center* também estão sujeitas à falhas, por isso, projetamos nossa arquitetura de forma a oferecer resiliência na comutação dos pacotes nos *switches* de núcleo. Mesmo sem manter nenhuma informação de estado em tabela, graças à inovadora abordagem utilizada em nossa arquitetura, cada pacote leva um Identificador de Rota Emergencial (IRE). Este identificador permite a rápida reação à falha sem trocas de mensagens ou protocolos de sinalizações, eliminando a sobrecarga tradicional dos mecanismos

de controle.

A RDNA amplia o roteamento na origem para suportar uma abordagem resiliente determinística baseado em um mecanismo de proteção de rede ao longo de todo o percurso. O IRE é calculado de forma a representar um conjunto de *switches* e suas respectivas portas de saída, necessárias para contornar uma falha na rota principal, assim, independentemente da posição da falha, conseguimos garantir a rápida reação à falha através de um caminho alternativo disjunto dentro da topologia de rede no *Data Center*.

Para ilustrar esse conceito, considere o cenário apresentado na Figura 3.3. Como no cenário anterior, uma VM "verde" localizada no $host_1$ deseja se comunicar com outra VM "verde" localizada no $host_{56}$. Então, os passos I, II e III são repetidos, mas é importante notar que, como parte do **Passo I**, o controlador RDNA também calculou o IRE como um valor único que compõe uma rota de proteção para toda a rota principal. Neste exemplo, temos $IRE = 117555379$, que define os *switches* $S = \{37,53,47,41,59\}$ (com suas portas de saída $p = \{15,1,13,15,3\}$) como a rota de proteção. Assim, tanto o IRP quanto o IRE precisam ser incorporados nos pacotes de entrada pelo *switch* de borda (posicionado no $host_1$).

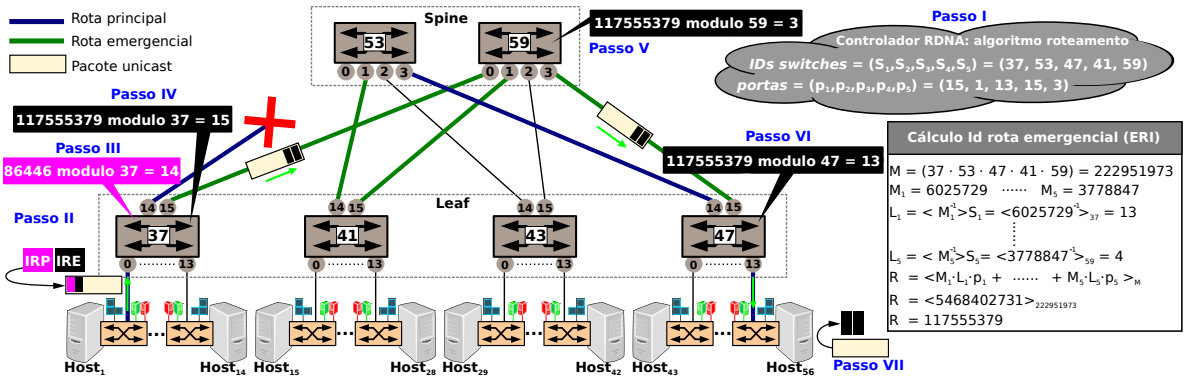


Figura 3.3: Roteamento RDNA unicast: rápida reação a falha.

Agora, nesse cenário, há um enlace em falha entre S_{37} e S_{53}. Antes do encaminhamento de pacotes, o *switch* S_{37} verifica o enlace de conexão. Como não está disponível, ele deve operar a partir do IRE e não mais pelo IRP. Uma opção é sobrescrever o IRP atual com o IRE. Então, usando o IRE, S_{37} calcula novamente o resto da divisão para poder encaminhar o pacote corretamente. Neste exemplo, usando o ID da rota ($R = 117555379$), S_{37} encaminha o pacote para a porta $< 117555379 >_{37}= 15$ (**Passo IV**). Quando S_{59} recebe um pacote com ID de rota ($R = 117555379$), ele encaminha o pacote para a porta $< 117555379 >_{59}= 3$ (**Passo V**). Então, quando S_{47} recebe um pacote com ID de rota ($R = 117555379$), ele encaminha o pacote para a porta $< 117555379 >_{47}= 13$ (**Passo VI**). Observe que substituindo o IRP pelo IRE os *switches* de núcleo S_{59} e S_{47} não precisam ser notificados da falha, nem estarem cientes que o pacote vem de um redirecionamento, portanto, para esses *switches* o pacote será encaminhado usando o identificador de rota

principal, ou seja, utilizando o IRP que foi substituído anteriormente pelo *switch* de núcleo que detectou a falha. Finalmente, o pacote atinge o *switch* de borda de egresso, que remove os IDs de rotas do pacote (**Passo VII**) e os entrega para a VM "verde" no $host_{56}$, que é o destinatário *dst*.

Como mencionado anteriormente, essa abordagem permite recuperação rápida para toda a rota principal. Por exemplo, se ocorrer uma falha no enlace entre S_{53} e S_{47}, S_{53} executará o **Passo IV**, sobrescrevendo o IRP e encaminhando o pacote para a porta $< 117555379 >_{53} = 1$. Quando S_{41} recebe um pacote com ID de rota ($R = 117555379$), ele encaminha o pacote para a porta $< 117555379 >_{41} = 15$. Então, quando S_{59} recebe um pacote com ID de rota ($R = 117555379$), ele encaminha o pacote para a porta $< 117555379 >_{59} = 3$ (**Passo V**). Finalmente, quando S_{47} recebe um pacote com ID de rota ($R = 117555379$), ele encaminha o pacote para a porta $< 117555379 >_{47} = 13$ (**Passo VII**), permitindo que o pacote alcance o *switch* de borda no $host_{56}$, que remove o ID da rota do pacote (**Passo VII**) e o entrega para a VM "verde" *dst* de forma transparente.

Uma outra forma de realizar a reação rápida é considerar no pacote um campo que indique ao *switch* de núcleo o modo de operação do encaminhamento, ou seja, antes de efetuar o encaminhamento, cada *switch* lê no cabeçalho do pacote a indicação de qual identificador de rota ele deve utilizar. Vamos considerar um campo no cabeçalho com valor 0, esta informação indica que o *switch* deve utilizar o IRP, o mesmo campo no cabeçalho com o valor 1 significa que o *switch* deve utilizar o IRE, com o valor 2 indica que é um pacote multicast e deve utilizar além dos resíduos uma expressão polinomial específica. Assim, se um *switch* for notificado sobre uma falha em uma porta na qual deveria encaminhar o pacote, ele poderia modificar o valor deste campo no cabeçalho do pacote, do valor 0 para o valor 1 e operar a partir desta nova informação. Em seguida, o *switch* do próximo salto saberá que deve utilizar o IRE e não o IRP. Este recurso é interessante, principalmente para implementações em *hardware* do tipo *cut-through*, pois se o resíduo apontar para uma porta não disponível o *switch* pode modificar o valor do campo no cabeçalho e já posicionar o pacote na rota emergencial. Esta ação permite reduzir a latência na comutação dos pacotes mesmo em períodos transitórios de falha, permitindo que o *switch* execute o encaminhamento de forma independente, mantendo a rede operante com eficiência.

Outro ponto importante que vale ressaltar é que, a arquitetura RDNA realmente não requer que os *switches* de núcleo sejam habilitados com SDN tradicional através do protocolo OpenFlow. A operação do módulo, o cálculo da expressão polinomial, a substituição do IRP pelo IRE ou a leitura do campo que indique o identificador de rota que o *switch* deve operar são, de fato, as principais funções a serem suportadas. No entanto, a SDN canônica permitiria que os *switches* de núcleo notificassem os controladores sobre falhas adicionais, por exemplo, falhas na rota emergencial. A SDN também forneceria suporte ao registro e a reconfiguração dinâmica dos IDs dos *switches*.

3.6 Formulação de Proteção Completa

Para proteção completa do fluxo ao longo de toda rota principal, o Controlador RDNA deve considerar rotas alternativas para cada possível falha de enlace ou *switch* da rota principal. Em uma arquitetura Clos de duas camadas (*Spine* & *Leaf*), o número de *uplinks* dos *switches Leaf*, l é igual ao número de *switches Spine*, v. Assim, o número total de conexões físicas é o número de *switches Leaf*, l multiplicado pelo número de *switches spine*. Portanto, cada *switch* de camada inferior é conectado a cada *switch* de camada superior formando uma topologia de malha completa (*full mesh*) [He et al. 2015]. Esta característica permite ao controlador RDNA explorar rotas emergenciais disjuntas em v de forma a proteger todo caminho principal.

De volta à Figura 3.3, onde a rota principal é $S = \{37,53,47\}$, as rotas alternativas são $S = \{37,59,47\}$ e $S = \{53,41,59,47\}$, para as falhas nos enlaces $S_{37}{\rightarrow}S_{53}$ e $S_{53}{\rightarrow}S_{47}$, respectivamente. Portanto, o Controlador RDNA deve realizar a união ($\cup$) dos identificadores dos *switches* que compõem essas rotas, resultando em $S = \{37,41,47,53,59\}$. Assim, o IRE é calculado simplesmente considerando os *switches* da rota principal e os *switches* adicionais que compõem as rotas emergenciais (alternativas), mas usando diferentes portas para cada *switch* de núcleo (S_i). Devido a união dos identificadores de S, podemos afirmar que o tamanho de S é delimitado pelo número de *switches* (S_i), e não pelo número de rotas alternativas que são inseridas.

O uso de um único identificador de rota emergencial fornece como vantagem um mecanismo simplificado de proteção. Este mecanismo permite ao *switch* de núcleo a rápida recuperação ao longo de toda a rota principal. A geração do IRE é baseada na rota principal que se deseja proteger. Inspirado no algoritmo de Suurballe [Abramowitz 1974], nos consideramos os seguintes procedimentos para geração do identificador da rota emergencial.

1. Eliminar as arestas $S_{37} \rightarrow S_{53}$ e entre $S_{53} \rightarrow S_{47}$ no grafo G, utilizadas na rota principal (IRP), isso vai gerar um grafo G' modificado;

2. Atualizar o custo das arestas de G' de acordo com:
   Os pesos em w considerando a soma dos identificadores entre os pares de S_i diretamente conectados.

3. Obter um novo caminho mínimo (src, dst) no grafo modificado;

4. Gerar um subgrafo unindo os 2 caminhos mínimos no grafo modificado.

Desse modo, o subgrafo resultante é composto de um caminho arco-disjunto entre (src, dst) em G. Em [Suurballe and Tarjan 1984] prova-se também que esse par tem soma mínima. A Figura 3.4 ilustra um exemplo de execução do nosso algoritmo adaptado,

usado para obter um caminho arco-disjunto de uma VM localizada no $host_1$ para outra VM localizada no $host_{56}$, seguindo as ilustrações anteriores.

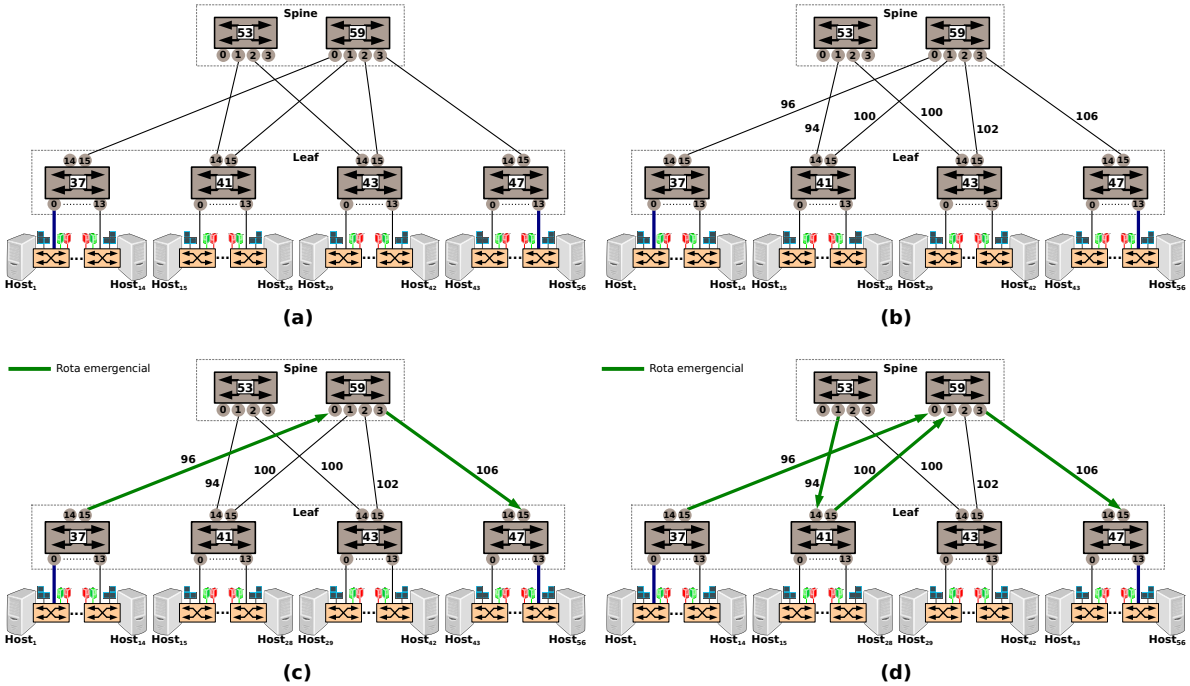


Figura 3.4: Exemplo do funcionamento do algoritmo para geração do IRE com proteção completa da rota principal. Em **(a)** remove-se as arestas do caminho principal gerando um grafo G'. Em **(b)** ilustra-se os pesos dos enlaces entre os *switches* do núcleo. Em **(c)** o menor caminho é encontrado. Em **(d)** o menor caminho entre (S_2,dst), cuja união ($\cup$) resulta no Identificador de Rota Emergencial (IRE).

Observe que na Figura 3.4 ao remover a aresta entre $S_{37} \rightarrow S_{53}$ e $S_{53} \rightarrow S_{47}$ o algoritmo garante que a rota emergencial não irá compartilhar nenhum enlace com a rota principal (Etapa 1). Se for aplicado um algoritmo qualquer de menor caminho como Dijkstra ou Bellman-ford [Bellman 1958], já seria possível determinar um caminho mínimo disjunto entre src e dst. Entretanto, a otimalidade só é garantida se utilizada também a atualização dos pesos das arestas, feita na Etapa 2.

O próximo passo é encontrar o caminho mínimo entre src e dst. Neste exemplo, ilustrado na Figura 3.4 é $S_{37} \rightarrow S_{59}$ e $S_{59} \rightarrow S_{47}$ compõe a rota emergencial. Observe que apesar de existir um único enlace possível os pesos adicionados na etapa anterior vão garantir a condição de otimalidade para topologias maiores (Etapa 3).

De posse de uma rota disjunta a partir de src, o algoritmo precisa encontrar um caminho cuja soma das arestas seja mínima entre S_2 até dst. Então, retornando a Etapa 3, o algoritmo encontrará um caminho mínimo (S_2, dst), que não será disjunto em relação a rota emergencial. Esta situação é desejada, tendo em vista que a condição de otimalidade irá garantir o reuso do mesmo identificador de *switch* de núcleo (S_{59}), que já faz parte da rota disjunta definida anteriormente. A Etapa 4 consiste em gerar um subgrafo com a união das rotas emergenciais $src \rightarrow dst$ e $S_2 \rightarrow dst$. Por fim, cabe ao Controlador RDNA

calcular o IRE considerando os *switches* do subgrafo com suas respectivas portas de saída.

O custo do nosso algoritmo adaptado é o de se encontrar a árvore mínima enraizada em *src*, mais duas execuções adicionais do algoritmo de Dijkstra para cada rota de proteção completa desejada. Isso ignorando os custos envolvidos nas Etapas 1 e 2, que podem ser reduzidos significativamente com implementações eficientes. Dada nossa estrutura topológica podemos afirmar que o custo computacional permanece inalterado para encontrar rotas emergenciais para qualquer instância de VM ou *container* em execução no DC.

3.7 Codificação RDNA para Comunicação Multicast

A abordagem da RDNA para comunicação multicast constrói a árvore multicast com base na codificação do *bitmap* que representa o conjunto de portas para o qual os pacotes devem ser encaminhados (em cada *switch* de núcleo). No entanto, o problema fundamental neste caso é como selecionar os pares de números primos para S_i, de modo que cada S_i deve ser maior que o número de combinações possíveis de portas de saída para construir a árvore multicast desejada. Por exemplo, considere uma topologia de rede de 2 camadas no DC, onde $Spine = 2$ e $Leaf = 4$, com 16 portas por *switch*. Nesse caso, devemos encontrar 6 números primos maiores que 2^{15} (a porta de entrada não precisa ser incluída).

Em vez de usar a operação de módulo para calcular diretamente o *bitmap* que indica as portas de saída para os pacotes, conforme utilizado em [Jia 2014], nossa arquitetura inova ao utilizar a operação de módulo para obter os valores de coeficiente para uma função polinomial específica. Uma abordagem autêntica que reduz consideravelmente a sobrecarga no cabeçalho, comparada com COXcast [Jia 2014], além de manter toda compatibilidade com nosso mecanismo de encaminhamento unicast.

Para explicar como nossa abordagem multicast funciona, considere a Figura 3.5. Imagine que exista uma máquina virtual "verde" (VM) instanciada em cada um dos 56 (cinquenta e seis) *hosts* físicos (não incluídos na figura por questões de clareza). Neste exemplo, a VM "verde", localizada no $host_1$ deseja efetuar uma transmissão multicast apenas para o grupo pertencente a um inquilino específico. Vamos considerar que este grupo multicast é formado por todas as VMs da mesma cor. A VM localizada no $host_1$ gerou um pacote e deseja transmitir cópias deste pacote para todos os membros de seu grupo em uma rede de núcleo sem estado. Do mesmo modo descrito anteriormente, o controlador RDNA calcula os identificadores, que em seguida são enviados para os *switches* de borda (**Passo I**), apresentado na mesma figura.

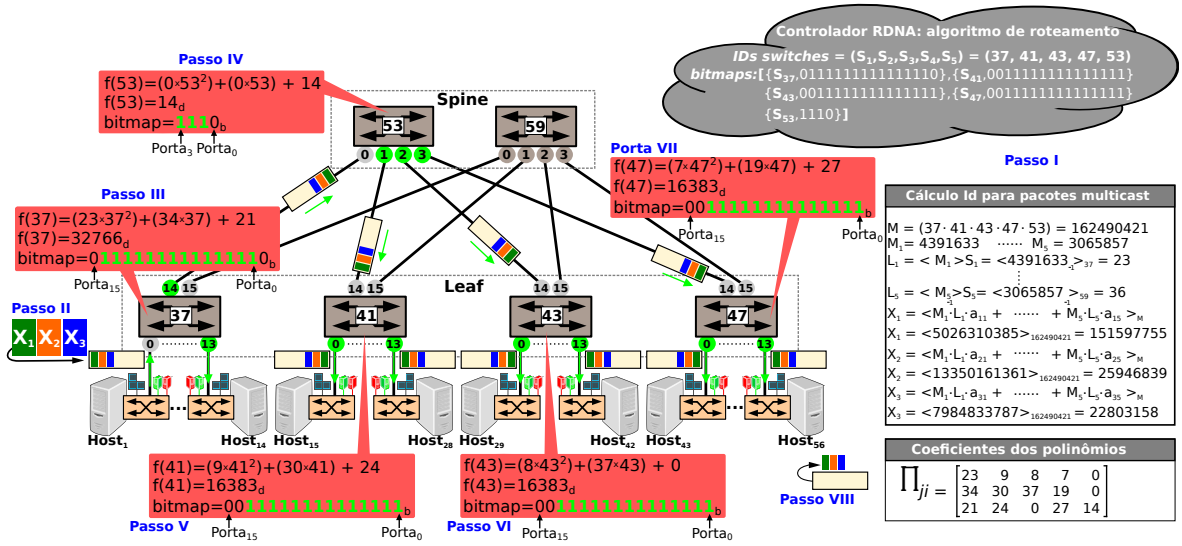


Figura 3.5: RDNA multicast com roteamento na origem utilizando resíduos e uma expressão polinomial algébrica específica sobre uma topologia 2-tier Clos Network.

Para a configuração topológica ilustrada na Figura 3.5, considere que três identificadores multicast, denotados por $X_1 = 151597755$, $X_2 = 25946839$ e $X_3 = 22803158$, foram calculados como parte do **Passo I**. No **Passo II**, estes identificadores são codificados no pacote, que é então encaminhado do *switch* de borda ($host_1$) para o primeiro *switch* de núcleo S_{37}. Como parte do **Passo III**, o *switch* executa a operação de módulo entre X_i e S_{37}, obtendo $< 151597755 >_{37} = 23$, $< 25946839 >_{37} = 34$ e $< 22803158 >_{37} = 21$. Em seguida, calcula a função polinomial $f(37) = (23 \times 37^2) + (34 \times 37) + 21 = \mathbf{32766}$, que corresponde ao *bitmap* (0111111111111110_b), especificando as portas de saída para encaminhar o pacote.

Ainda na Figura 3.5 observe que todos os *hosts* conectados a S_{37} (portas 1 até 13), receberão uma cópia do pacote, exceto a porta 0 que é a porta de origem. A porta 14 também receberá uma cópia do pacote. Para simplificar a explicação mantivemos o bit "1" na cor verde indicando que a respectiva porta de saída receberá uma cópia do pacote.

Quando o pacote chega em S_{53} (Figura 3.5), o *switch* de núcleo lê o cabeçalho e calcula o módulo com o mesmo X_1, X_2, X_3 para descobrir os valores dos coeficientes, obtendo $< 151597755 >_{53} = 0$, $< 25946839 >_{53} = 0$ e $< 22803158 >_{53} = 14$ (**Passo IV**). Calculando a expressão polinomial, o *switch* de núcleo S_{53} obtém o valor decimal $f(53) = (0 \times 53^2) + (0 \times 53) + 14 = \mathbf{14}$, que resulta no *bitmap* (1110_b).

Quando o pacote multicast chega em S_{41}, S_{43} e S_{47} (Figura 3.5), o mesmo processo acontece, mas o *bitmap* deve ser (0011111111111111_b) para formar a árvore multicast desejada. Então, eles calculam o módulo com o mesmo X_1, X_2, X_3 obtendo seus próprios valores de coeficientes que são para S_{41} $< 151597755 >_{41} = 9$, $< 25946839 >_{41} = 30$, $< 22803158 >_{41} = 24$, por S_{43} são $< 151597755 >_{43} = 8$, $< 25946839 >_{43} = 37$, $< 22803158 >_{43} = 0$ e para S_{47} são $< 151597755 >_{47} = 7$, $< 25946839 >_{47} = 19$, $< 22803158 >_{47} = 27$ (**Passos V, VI e VII**).

Observe que em nenhum momento os identificadores de rota multicast ($X_1 = 151597755$, $X_2 = 25946839$ e $X_3 = 22803158$) foram alterados no cabeçalho. Da mesma forma que nenhum estado é requerido no núcleo da rede, cabe ao *switch* apenas realizar a operação de módulo e calcular a expressão polinomial, cujo resultado indicará um *bitmap* com a representação das portas de saída dos pacotes.

3.7.1 Calculando o grau da função polinomial

O primeiro passo para calcular a codificação multicast na arquitetura RDNA é determinar o grau do polinômio, que deve ser configurado e entregue na fase de inicialização, assim como o identificador S_i para cada *switch* de núcleo.

O Pseudocódigo 1 apresenta um algoritmo simples para obter o grau polinomial d. Esse algoritmo pesquisa o valor mínimo de d (a partir de 1) para que o resultado do polinômio seja suficiente para reproduzir toda a representação em bits (*bitmap*) válido para todos os *switches* do núcleo. Então, para cada *switch*, dado seu ID S_i e seu número de portas físicas p, calculamos o grau d usando como coeficientes o valor máximo possível do módulo para S_i, ou seja, $S_i - 1$) [1] (**Linha 2**). Por exemplo, no caso do *switch* de núcleo S_{37}, que tem 16 portas, o grau 2 satisfaz esta condição ($f(37) = (36 \times 37^2) + (36 \times 37) + 36 > 2^{15}$) (**Linha 9**).

Algorithm 1 Calculando o grau do polinômio algébrico.

```
1:  function DEGREE(S_i,p)
2:      fcoef ← S_i − 1;
3:      decimal ← 0;
4:      d ← 1                                          ▷ Grau do polinômio
5:      maximum ← 2^(p−1)
6:      Seek ← False
7:      while Seek = False do
8:          decimal ← decimal + (fcoef + (fcoef.(S_i^d)))
9:          if decimal ≥ maximum then
10:             Seek ← True
11:         else
12:             d ← d + 1
13:     return d                                       ▷ Grau do polinômio encontrado
```

Depois de obter o grau polinomial d, o controlador RDNA também precisa calcular os coeficientes polinomiais para cada árvore multicast a ser configurada na topologia. No caso do DC ilustrado na Figura 3.5, esses coeficientes são (a_3, a_2, a_1) de $f(i) = a_3 \times i^2 + a_2 \times i + a_1$. Eles serão usados por todos os *switches* para calcular o *bitmap* com a representação das portas de saída que receberão uma cópia do respectivo pacote.

[1] O teorema da divisão euclidiana declara que um resto r de uma computação de divisão é um inteiro tal que $0 \leq r < b$, onde b é o divisor [Gomes et al. 2016].

3.7.2 Calculando os coeficientes da expressão polinomial

O Pseudocódigo 2 mostra nosso algoritmo para descobrir os coeficientes para cada S_i. Na (**Linha 16**), o controlador RDNA tem como entrada o ID do *switch* de núcleo S_i, o grau do polinômio (*degree*) e o *bitmap* requerido. Por exemplo, o algoritmo recebe como entrada 37, 2 e 0111111111111110_b. Então, a partir do termo de maior grau (i^2) o algoritmo pesquisa o valor máximo de a_3, de forma que $(a_3 \times 37^2) \leq 32766_d$=$0111111111111110_b$. Nesse caso, o valor $23 \times 37^2 = 31487$ é menor que 32766, logo, 23 é adicionado na lista de coeficientes $coefvalues$.

Algorithm 2 Calculando os coeficientes da expressão polinomial.

```
1:  coefvalues = Null;                          ▷ Lista com os valores dos coeficientes in S_i
2:  function COEF(S_i,exponent,decimal)
3:      tmpvalue ← 0;
4:      limit ← (S_i − 1);                                  ▷ Limite para o coeficiente
5:      for c ∈ range(limit,-1, -1) do
6:          tmp ← c · (S_i^exponent);
7:          if decimal ≥ tmp then
8:              tmpvalue ← (decimal − tmp);
9:              Break;
10:     if tmpvalue ≥ 0 then
11:         coefvalues ← c;
12:     else
13:         coefvalues ← 0;
14:     return tmpvalue
15: function GET COEF(S_i, degree, bitmap)
16:     decimal ← int(bitmap, 2)          ▷ Converte o valor decimal para binário
17:     for exp ∈ range(degree,-1, -1) do
18:         if (decimal < S_i) then
19:             if (len(list) = degree) then
20:                 coefvalues ← decimal;
21:                 Break;
22:             else
23:                 coefvalues ← 0;
24:         else
25:             tempwish ← Coef(S_i,exp,decimal)
26:             if tempwish ≥ 0 then
27:                 decimal ← tempwish
28:     return coefvalues                  ▷ Retorna lista com os coeficientes
```

Para o próximo termo (i^1), seguimos a mesma lógica, mas usando 1279 (32766 − 31487) como limite. O algoritmo procura o valor máximo de a_2, de forma que $(a_2\times) \leq 1279$. Nesse caso, o valor 34, $(34 \times 37 = 1258)$, portanto, é adicionado à lista de coeficientes $coefvalues$, e o restante 21 $(1279 - 1258)$ é usado diretamente como o último coeficiente (i^0), terminando o algoritmo para os coeficientes em S_{37}.

O mesmo processo ocorre para S_{41}, S_{43}, S_{47} e S_{53}, e a matriz resultante é Π_{ji}, mostrada na Figura 3.5 (**Passo I**). As colunas representam as listas de coeficientes para o conjunto de *switches* de núcleo que compõem a árvore multicast: S_{37}, S_{41}, S_{43}, S_{47} e S_{53}, respectivamente.

O último passo *é calcular* X_i, baseado no Teorema Chinês do Resto [Ding et al. 1996] (TCR) como descrito anteriormente. No entanto, para a comunicação multicast na arquitetura RDNA, as portas de saída denotadas por p_i agora são substituídas pelos coeficientes do polinômio Π_{ji} que reproduz o *bitmap* necessário para S_i. Note que X_i representa o resíduo e L_i é o inverso multiplicativo modular de M_i. Esta ação termina com a codificação da RDNA no **Passo I** apresentada na Figura 3.5 como segue:

$$M = 37 \cdot 41 \cdot 43 \cdot 47 \cdot 53 = 162490421$$
$$M_1 = 4391633, M_2 = 3963181, M_3 = 3778847,$$
$$M_4 = 3457243, M_5 = 3065857$$
$$L_1 =< 4391633^{-1} >_{37}= 23,\ L_2 =< 3963181^{-1} >_{41}= 20$$
$$L_3 =< 3778847^{-1} >_{43}= 37,\ L_4 =< 3457243^{-1} >_{47}= 36$$
$$L_5 =< 3065857^{-1} >_{53}= 14$$
$$X_1 =< L_1 \cdot M_1 \cdot \Pi_{11} + L_2 \cdot M_2 \cdot \Pi_{12} + L_3 \cdot M_3 \cdot \Pi_{13}+ L_4 \cdot M_4 \cdot \Pi_{14} >_M + L_5 \cdot M_5 \cdot \Pi_{15} >_M$$
$$X_1 =< 4391633 \cdot 23 \cdot 23 + 3963181 \cdot 20 \cdot 9+ 3778847 \cdot 37 \cdot 8 + 3457243 \cdot 36 \cdot 7+ 3065857 \cdot 14 \cdot 0 >_M$$
$$X_1 =< 2323173857 + 713372580 + 1118538712+ 871225236 + 0 >_{162490421}= 151597755$$

3.8 Considerações Finais

O objetivo deste capítulo foi expor a visão conceitual da RDNA. Nossa arquitetura é idealizada a partir da distinção clara dos blocos funcionais (borda, núcleo e gerenciamento). Portanto, na Seção 3.1 descrevemos a concepção e as atribuições de cada bloco funcional. Em seguida, na Seção 3.2 explicamos o funcionamento do sistema de roteamento, incluindo o formalismo para geração das rotas utilizado Teorema Chinês do Resto.

Na Seção 3.3 descrevemos o método para geração do identificador único utilizado nos *switches* de núcleo. Em seguida, na Seção 3.4 explicamos a codificação e o funcionamento do modelo de encaminhamento dos pacotes baseado em resíduo para comunicação unicast. Como as redes de *Data Center* estão sujeitas a falhas, na Seção 3.5 descrevemos a codificação e o funcionamento do nosso robusto mecanismo de reação rápida à falha. Este mecanismo original foi projetado de tal forma que viabiliza a proteção de toda rota principal utilizando apenas um único identificador emergencial. Logo depois, na Seção 3.6 formalizamos um procedimento para codificar o identificador de rota emergencial, através de um algoritmo adaptado que garante o menor caminho na rota de recuperação, além de reduzir a sobrecarga (comprimento dos bits) no cabeçalho.

Finalmente na Seção 3.7, dissertamos sobre nosso inovador sistema de roteamento e encaminhamento multicast. Para este tipo de comunicação, a arquitetura RDNA utiliza

como base o Teorema Chinês do Resto em conjunto com uma expressão polinomial algébrica específica. A operação de resíduo determina o valor para cada coeficiente que é multiplicado pelo identificador do *switch* de núcleo (S_i). O somatório dos monômios do polinômio é convertido para um número binário que reproduz de forma eficiente um mapa de bits (*bitmap*) indicando as portas de saída do pacote. De forma complementar, também explicamos o funcionamento do algoritmo que determina o grau do polinômio, além do algoritmo que calcula o valor para cada coeficiente da expressão polinomial.

Capítulo 4

Análise de Escalabilidade

Neste Capítulo, apresentamos a análise de escalabilidade da arquitetura RDNA, ademais, comparamos a arquitetura RDNA com outras propostas existentes. Nossa avaliação considera as abordagens de comunicação unicast, incluindo o roteamento resiliente, além da comunicação multicast utilizando *switches* de núcleo sem tabela. A análise de escalabilidade é estruturada em três partes. A primeira parte, descrita na Seção 4.1, consiste na avaliação analítica da arquitetura RDNA, à partir dos princípios defendidos anteriormente.A segunda parte, descrita na Seção 4.2 relata sobre o mecanismo de proteção para recuperação de falhas. Na terceira parte, descrita na Seção 4.3, abordamos a escalabilidade da RDNA para comunicação multicast.

4.1 Comunicação Unicast

Para termos uma referencia topológica comparativa nós assumimos o mesmo conjunto de topologias usadas em [Jia 2014] que inclui 23 *fan-outs* diferentes de 2-tier Clos Network. Nós selecionamos a abordagem COXcast como concorrente, dada suas características similares à nossa arquitetura, das quais podemos destacar: i) uso do Teorema Chinês do Resto como mecanismo para definição da rota; ii) os *switches* de núcleo não requererem tabela; iii) suportar comunicação unicast e multicast; e iv) não requer reescrita do pacote a cada salto.

O comprimento de um caminho unicast é calculado entre dois *hosts* (origem e destino), considerando o caminho mais curto ($n = 3$). O número máximo de Bytes requerido pela codificação da arquitetura RDNA para unicast (R) pode ser calculado da seguinte forma:

$$R = \frac{\left(\log_2\left(M = \prod_{i=1}^{n} S_i\right)\right)}{8} \tag{4.1}$$

A Equação 4.1 indica que quanto maior o valor de M, maior o tamanho em *bits* máximo requerido. Lembrando que IRP é o ID da rota principal que varia em função do número de *switches* no caminho, n e dos IDs dos *switches* de núcleo S_i.

A Tabela 4.1 apresenta a escalabilidade da RDNA versus COXcast [Jia 2014]. Entretanto, apesar das similaridades na abordagem, como pode ser visto, a RDNA é significativamente mais escalável que o COXcast, reduzindo na média 4.5 vezes o tamanho do cabeçalho para a comunicação unicast. Em alguns casos, a redução é ainda maior, como de 35 para apenas 4 Bytes. A razão para isso é que o COXcast requer grandes números primos para IDs de *switches*, o que aumenta drasticamente a sobrecarga, em particular para algumas topologias com *switches* de até 96 portas. Maiores detalhes da arquitetura COXcast serão discutidos no Capítulo 6.

Tabela 4.1: Tamanho do cabeçalho (Bytes) para configurações típicas de um *Data Center* utilizando topologia 2-tier Clos Network.

Configurações 2-tier	**Portas**	**Hosts físicos**	**RDNA (IRP)**	**RDNA (IRE)**	**COXcast**
Spine = 02 Leaf = 04	16	56	2	4	5
	24	88	4	7	8
	32	120	2	4	11
Spine = 04 Leaf = 08	16	96	2	4	5
	24	160	3	5	8
	32	224	3	4	11
	48	352	3	5	17
	96	736	3	5	35
Spine = 06 Leaf = 12	16	120	3	5	5
	24	216	3	5	8
	32	312	3	5	11
	48	360	4	6	17
	96	1080	4	6	35
Spine = 12 Leaf = 16	16	160	3	5	5
	24	288	3	5	8
	32	416	3	5	11
	48	672	4	6	17
	96	1440	4	6	35
Spine = 08 Leaf = 16	16	128	3	5	5
	24	256	3	5	8
	32	384	3	5	11
	48	640	4	6	17
	96	1408	4	6	35

4.2 Comunicação Unicast Resiliente

Nossa análise pressupõe uma proteção completa de toda a rota principal, assim, qualquer falha que ocorra em qualquer ponto ao longo da rota principal (IRP), fará com que o tráfego seja movido para a rota emergencial (IRE), até que a rota principal seja reestabelecida. Tomando novamente a Figura 3.3, considere que uma VM qualquer no $host_1$ é src e outra VM localizada no $host_{56}$ é o dst. IRP é definido usando os *switches* de núcleo $S = \{37,53,47\}$ com portas de saída $p = \{14,3,13\}$ e o IRE é definido usando os *switches* $S = \{37,53,47,41,59\}$ com portas de saída $p = \{15,1,13,15,3\}$.

Conforme mostrado na Tabela 4.1, a RDNA é capaz de suportar a recuperação de falhas para proteger toda a rota principal com tamanho de cabeçalho variando de 3 $\sim$ 7 Bytes para todas as topologias avaliadas. Essa proteção total é o pior cenário para a codificação RDNA IRE, que corresponde ao número máximo de *Bytes* requeridos pelo IRE (ver Equação 4.1).

Para proteção total usada pelo IRE, o controlador RDNA precisa encontrar os caminhos disjuntos com a contagem mínima de saltos. Assim, nós buscamos como inspiração um algoritmo chamado Suurballe [Abramowitz 1974, Girolimetto et al. 2017], e adaptamos uma solução cujo tempo computacional é duas vezes o tempo de Dijkstra [Dijkstra 1959]. Conforme apresentamos anteriormente na Seção 3.6, nosso algoritmo permite usar os mesmos *switches* $Leaf$ e *Spine*, desde que eles tenham o mesmo *host* final como destino, através da união ($\cup$) de *switches* comuns (S_i) na rota emergencial. Portanto, se ocorrer uma falha no enlace entre S_{37} e S_{53}, S_{37} usará a porta 15 (IRE), $S_{37} \rightarrow S_{59}$, $S_{59} \rightarrow S_{47}$ e $S_{47} \rightarrow dst$, que corresponde a uma das rotas emergenciais codificadas no IRE ($S = \{37,59,47\}$). Se ocorrer uma falha no enlace entre S_{53} e S_{47}, S_{53} usará a porta 1, $S_{53} \rightarrow S_{41}$, $S_{41} \rightarrow S_{59}$, $S_{59} \rightarrow S_{47}$ e $S_{47} \rightarrow dst$, que corresponde a outra rota emergencial codificada no IRE ($S = \{37,53,41,59,47\}$).

Finalizando a análise de escalabilidade unicast com resiliência, observe na Tabela 4.1 que a codificação dos identificadores de rota RDNA se ajustam aos 12 *Bytes* do endereço MAC (*Media Access Control*) do quadro Ethernet. O IRP e o IRE podem ser incorporados no pacote (por exemplo, em campos de cabeçalho como: MAC de destino e MAC de origem), portanto, seu comprimento em termos de bits não afeta o *payload* do pacote, além de não gerar nenhum significativo *overhead* na incorporação dos identificadores de rota RDNA nos pacotes. Assumindo que o padrão Ethernet no núcleo é essencialmente para fins de enquadramento (*framing*), a RDNA pode usar os bits específicos e seu significado de acordo com as propriedades matemáticas do TCR. Por outro lado, o COXcast não descreve nenhuma recuperação de falha, por isso, não foi incluído em nossa análise.

4.3 Comunicação Multicast

Para a comunicação multicast, o nosso foco está em analisar e entender como o cabeçalho RDNA multicast é afetado pelo tamanho da rede e pelo tamanho do grupo multicast (quantidade de membros do grupo). Nos implementamos a simulação da análise de escalabilidade em Python para a comunicação multicast, cobrindo 23 (vinte e três) diferentes configurações de topologia 2-tier Clos Network. Dado que as topologias de rede 2-tier Clos são as topologias mais populares nas redes de *Data Center* [Alizadeh et al. 2014], nós a consideramos como referência para o modelo de simulação.

4.3.1 Impacto sobre o tamanho da rede

O multicast na RDNA foi concebido baseado sobre funções polinomiais que possuem um compromisso (*trade-off*) ótimo entre o conjunto e números primos S_i e o grau do polinômio d. Com o objetivo de encontrar o valor ótimo, ou seja, o menor tamanho de cabeçalho, nós formulamos um modelo de programação linear, como segue:

$$
\begin{aligned}
&\text{minimize} \quad X_j^{d+1} = \frac{\left(\log_2\left(\prod_{i=1}^{n} S_i\right)\right)}{8} \cdot \left(d+1\right) \\
&\text{minimize} \quad S_i \\
&\text{subject to} \\
&\qquad \sum_{d=1}^{m} \left(S_i - 1 + \left(S_i - 1 \cdot \left(S_i^d\right)\right)\right) \geq 2^{p_i - 1} \\
&\qquad i, j, d \geq 1, \quad m \leq 14 \\
&\qquad S_i \in \mathbb{N} \quad | \quad i \quad primo > p_i,
\end{aligned}
\tag{4.2}
$$

o objetivo desta função multi-objetivo é minimizar o tamanho do cabeçalho, considerando, X_j^{d+1} e S_i. Os resíduos X_j^{d+1} são sujeitos à $\sum_{d=1}^{m}$ que precisa ser maior ou igual do que o valor máximo para representação dos *bitmaps* 2^{p_i-1}, isto é, maior do que o número de portas físicas de saída p_i do *switch* S_i. Finalmente, nós limitamos o grau máximo do polinômio m para 14, colocando assim um limite no tamanho do cabeçalho da arquitetura RDNA.

A Figura 4.1 ilustra o processo de otimização, pegando como exemplo a maior configuração de rede dentre as 23 (vinte e três) configurações avaliadas, que é $Spine = 08$, $Leaf = 16$, com $porta = 96$, limitando o eixo x a 50k (50021). Para tal topologia de rede, isso significa que temos que selecionar 24 (vinte e quatro) S_i. No entanto, quanto maior for S_i ($\uparrow$), menor é o grau polinomial d ($\downarrow$). Então, se formos da esquerda para a direita, o tamanho dos números primos aumenta à medida que o grau polinomial diminui. Observe que há dois conjuntos mínimos de 24 números primos que requerem 200 Bytes para o cabeçalho multicast RDNA. A primeira opção começa em 727 [727, 733, 739, 743, 751,

757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881], destacado no quadrado rosa e um segundo conjunto começa em 3761 [3761, 3767, 3769, 3779, 3793, 3797, 3803, 3821, 3823, 3833, 3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907, 3911, 3917, 3919, 3923, 3929, 3931], ilustrado na seta rosa. Dado que o primeiro conjunto S_i é menor, então eles serão usados pelo controlador RDNA para atribuir aos *switches* de núcleo seus respectivos IDs.

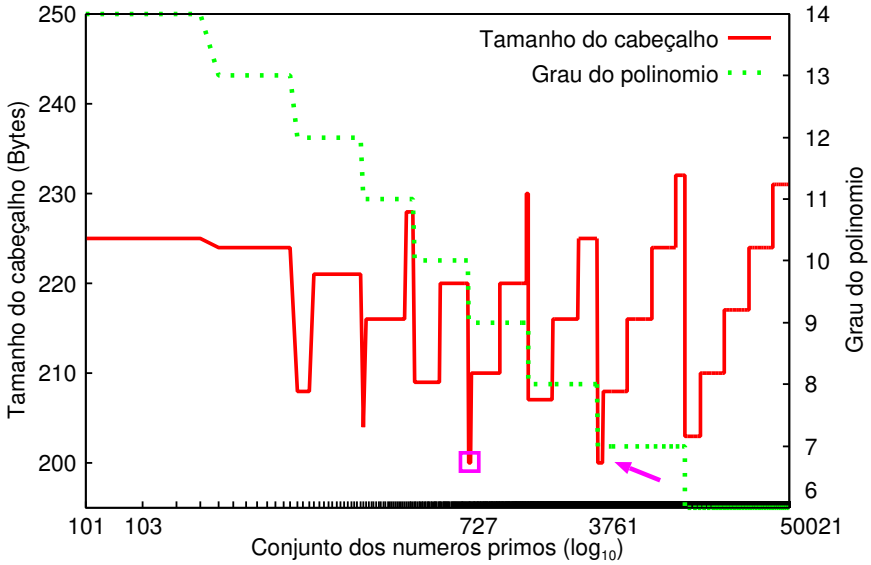


Figura 4.1: Análise do conjunto de números primos para a configuração $Spine = 08$, $Leaf = 16$, com $portas = 96$.

Após o processo de otimização, comparamos nossa arquitetura com os resultados apresentados em [Jia 2014], que inclui COXcast, Xcast4 e Xcast6 [Jia 2014], para vários tamanhos de redes 2-tier Clos Network, variando do $Spine = 2$, $Leaf = 4$ com $portas = 16$ até $Spine = 8$, $Leaf = 16$ com $portas = 96$.

Conforme apresentado na Tabela 4.2, podemos observar que na medida que aumentamos o tamanho da rede, naturalmente, o cabeçalho utilizado para representar a árvore multicast também aumenta, contudo o incremento é bem dimensionado. Ocasionalmente, chegando em até 200 Bytes para os piores casos avaliados, mas na maior parte é inferior a 100 Bytes. Além disso, o multicast da RDNA reduz o tamanho do cabeçalho em todos os casos em comparação com o Xcast4 e o Xcast6. Em média, a RDNA reduz o tamanho do cabeçalho em 12% em comparação com o COXcast, mas pode atingir 50% para alguns casos, por exemplo, em $Spine = 8$, $Leaf = 16$ com $portas = 16$. Isso se deve ao fato de que o COXcast precisa ter números primos enormes para representar a chave "*Key*" específica do nó de tal forma que o resto da divisão resulte nas portas de saída como um *array* de bits não inteiros. A diferença entre COXcast e RDNA é significativamente maior sempre que o número de portas (grau de conectividade) é baixa. Por exemplo, para topologias em que os *switches* têm 16 portas, a arquitetura RDNA solicita em torno de 27% menos Bytes do que o COXcast.

Tabela 4.2: Sobrecarga (*overhead*) máximo em Bytes versus tamanho da rede.

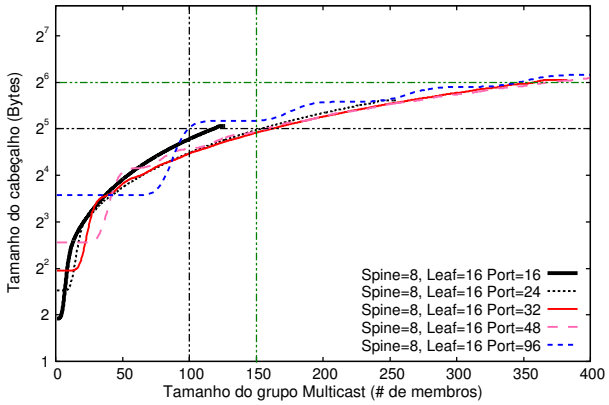
Configuração 2-tier	Portas	RDNA	COXcast	Xcast4	Xcast6
Spine = 02 Leaf = 04	16	9	10	12	1.008
	24	14	14	380	1.520
	32	18	18	508	2.032
Spine = 04 Leaf = 08	16	18	22	508	2.032
	24	27	30	764	3.056
	32	35	38	1.020	4.080
	48	54	54	1.532	6.128
	96	104	106	3.068	12.272
Spine = 06 Leaf = 12	16	26	36	764	3.056
	24	39	48	1.148	4.592
	32	52	60	1.532	6.128
	48	75	84	2.300	9.200
	96	154	156	4.604	18.416
Spine = 06 Leaf = 16	16	34	47	1.020	4.080
	24	51	63	1.532	6.128
	32	68	79	2.044	8.176
	48	100	111	3.068	12.272
	96	200	207	6.140	24.560
Spine = 08 Leaf = 16	16	34	51	1.020	4.080
	24	51	67	1.532	6.128
	32	68	83	2.044	8.176
	48	100	115	3.068	12.272
	96	200	211	6.140	24.560

4.3.2 Impacto sobre o cabeçalho considerando a quantidade de membros do grupo multicast

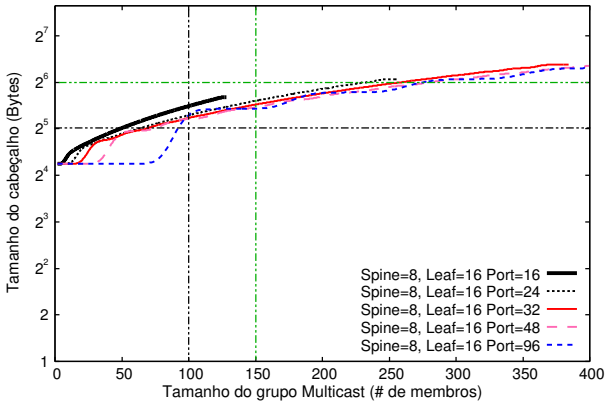
Considerando que um micro DC tem uma distribuição não esparsa das VMs (ou *containers*) em seus servidores físicos [Martini et al. 2015], assumimos que os membros do grupo multicast são distribuídos como *containers*/VM colocados em cada *host* físico. Assim, o tamanho do grupo multicast é incrementado sequencialmente, um por um até alcançar todos os *hosts* em cada configuração de rede.

Continuando nosso exemplo de uma comunicação RDNA multicast, ilustrado na Figura 3.5, a análise começa com uma VM qualquer localizada no $host_1$ origem. Então outra VM que pertence ao mesmo inquilino, localizada no $host_2$, é adicionada como membro do grupo, seguindo para outra VM localizada no $host_3$, e assim por diante até que todas as VMs que pertencem ao respectivo inquilino tenham sido incluídos como membros do

grupo. Os resultados são representados nas Figuras 4.2.1 e 4.2.2 para a maior configuração de rede.



(4.2.1) *multicast RDNA* .



(4.2.2) *COXcast.*

Figura 4.2: Cabeçalho multicast requerido, variando o número de portas para configuração 2-tier Clos Network.

A Figura 4.2.1 mostra que o tamanho do cabeçalho RDNA é escalável, considerando um número variável de portas físicas (16 até 96) com $Spine = 8$ e $Leaf = 16$. Note que se o tamanho do grupo for definido como 100 membros, o tamanho do cabeçalho será menor que 35 Bytes para todas as configurações. Aumentando o tamanho do grupo para 150 membros, somente a configuração com 96 portas é maior que 32 Bytes (2^5). Em outras palavras, o aumento de 50% nos membros multicast leva apenas a um pequeno incremento no tamanho do cabeçalho. Por exemplo, se aumentarmos a quantidade de membros do grupo para 350, o tamanho do cabeçalho ficará abaixo de 64 Bytes, mas agora com 3.5 vezes mais membros no grupo multicast. Incrementos maiores ocorrem toda vez que um *switch Spine* precisa ser adicionado para permitir a comunicação com uma nova folha ($Leaf$) no grupo multicast.

Além disso, a RDNA reduziu substancialmente o tamanho do cabeçalho, quando comparamos com o COXcast, conforme apresentado na Figura 4.2.2. Note que com o tamanho do grupo menor que 100 membros, o cabeçalho RDNA multicast se mantém menor que 2^4 Bytes para um grupo de até 55 membros, enquanto o COXcast precisa de 2^5 Bytes para todas as configurações. Incrementos maiores no tamanho do cabeçalho ocorrem para graus de nó mais altos, por exemplo, um *switch* $Leaf$ com 96 portas permite 88 conexões com *hosts* físicos $(96 - 8)$ [1] no grupo multicast a partir de um mesmo *switch Spine*. Portanto, a cada inclusão de um novo *switch Spine* temos um acréscimo no tamanho do cabeçalho multicast na arquitetura RDNA.

Resumindo, os resultados evidenciam que o multicast de RDNA é mais escalável do que a família Xcast, de modo que a arquitetura RDNA tem o potencial de suportar as

[1] $Tamanhodogrupo = [(p - v) \times l] - 1$ em que, p é o número de portas físicas em cada switch de folha ($Leaf$), v é o número de *switches Spine*, l é o número de *switches* $Leaf$, menos a origem.

demandas emergentes de projetos para pequenos *Data Centers* construídos com topologias 2-tier. Além disso, vimos que o cabeçalho multicast RDNA é claramente mais sensível ao tamanho da rede (especialmente quando os graus dos nós são altos) do que aos membros do grupo multicast, assumindo que a distribuição do grupo multicast não é esparsa.

4.4 Considerações Finais

Neste capítulo apresentamos a análise de escalabilidade da arquitetura RDNA considerando dois modelos de comunicação: unicast e multicast. Focamos nossa análise considerando a topologia 2-tier Clos Network, especialmente porque essa topologia é comumente encontrada em DC que suportam dezenas de milhares de servidores físicos [Alizadeh et al. 2014, Oueis et al. 2014, Mach and Becvar 2017].

Na Seção 4.1 analisamos a escalabilidade para comunicação unicast sem proteção, em seguida na Seção 4.2 discutimos a comunicação unicast considerando o impacto da rota de proteção. Em todos os casos a RDNA apresentou-se mais eficiente quando comparada com algumas propostas concorrentes, requerendo um comprimento de cabeçalho menor. Outra vantagem da RDNA foi manter o *frame* Ethernet inalterado, considerando diferentes arranjos topológicos. Por fim, na Seção 4.3 avaliamos o cabeçalho para comunicação multicast. Primeiro, considerando a variação de tamanho da rede (quantidade de *switches* na árvore multicast), e em seguida, o impacto no cabeçalho variando a quantidade de membros por grupo multicast. Também neste tipo de comunicação a RDNA se mostrou eficaz, reduzindo em até 50% o tamanho do cabeçalho multicast para alguns dos casos avaliados.

Capítulo 5

Prova de Princípios

Este capítulo descreve a implementação dos protótipos experimentais e a validação metodológica da nossa arquitetura. Também apresentamos e discutimos os resultados experimentais obtidos.

5.1 Implementação da RDNA

Com base nos conceitos descritos no Capítulo 3, ilustramos na Figura 5.1 o projeto de implementação da nossa arquitetura.

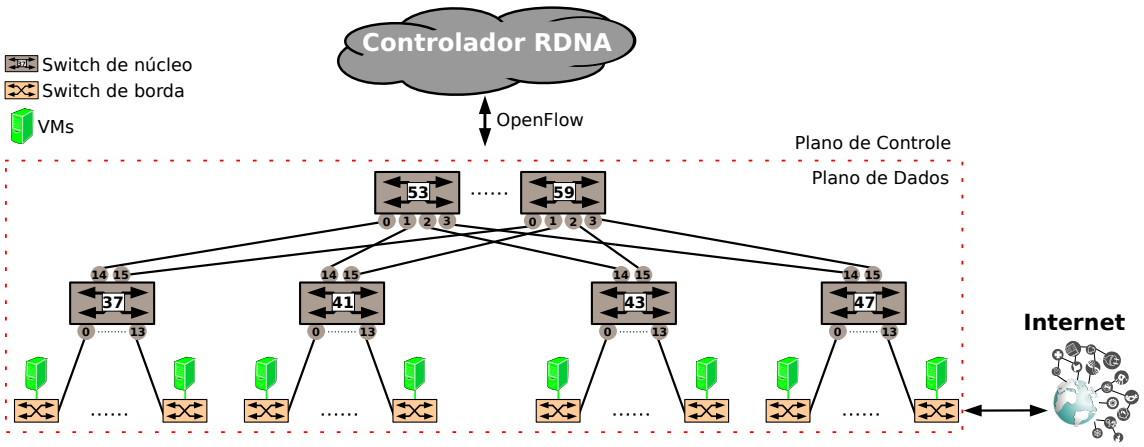


Figura 5.1: Visão geral da implementação da arquitetura considerando o controlador RDNA, e os *switches* de borda e núcleo. Em nossa arquitetura, um ou mais *switches* de borda podem ser conectados com redes de outros domínios (*gateway*).

Dado que um *Data Center* é na maioria das vezes, gerenciado por uma única entidade administrativa que detém o controle dos equipamentos e do *software* utilizado [Benson et al. 2010], isso leva à adoção de um controlador logicamente centralizado que toma decisões programáveis dentro deste ambiente.

5.1.1 Controlador de Rede

É uma entidade central responsável por configurar, monitorar e resolver problemas na infraestrutura do DC. Deve trabalhar de forma autônoma, auxiliando os administradores na configuração dos dispositivos de rede e na aplicação das políticas de gerenciamento local. Como plataforma habilitadora, consideramos o uso do arcabouço SDN, através de uma aplicação desenvolvida no Ryu [Ryu 2014] para apoiar o Controlador RDNA. O Ryu foi selecionado por ser totalmente implementado em Python, possui boa integração com outras ferramentas de rede, como por exemplo o OpenStack [Corradi et al. 2012], além de ser um projeto com suporte a vários protocolos de *southbound*, como OpenFlow, NetConf e OF-Config. O Ryu possui uma excelente comunidade de desenvolvimento, além de implementar por completo as versões 1.0, 1.2, 1.3, 1.4, 1.5 do OpenFlow e extensões da Nicira.

No controlador RDNA consideramos as seguintes funções: i) descoberta usando o *Link Layer Discovery Protocol* (LLDP); ii) define o caminho dos fluxos unicast e multicast utilizamos o algoritmo de roteamento Dijkstra, para a rota emergencial considerando a implementação do nosso algoritmo adaptado conforme apresentado no Capítulo 3; iii) instalação dos fluxos acontece de forma ativa e reativa através de mensagens de controle OpenFlow, *flow-mod* e *packet-in*, respectivamente; e iv) entrega do identificador e do grau do polinômio para os *switches* de núcleo, que foi realizada via mensagens do OpenFlow de extensão da Nicira, conforme realizado em [Vassoler 2015].

5.1.2 *Switches* de borda

Este elemento, recebe do Controlador RDNA os IDs das rotas. É importante notar que a funcionalidade do elemento de borda pode ser construída em *software*, ou seja, em vSwitchs. Optou-se em utilizar como elemento de borda o Open vSwitch (OvS) habilitado para OpenFlow 1.3 [vSwitch 2014].

Outra vantagem da utilização de um *switch* em *software* é que as funcionalidade podem ser alteradas com maior flexibilidade do que as implementações de *switches* em *hardware* convencional. O respectivo caminho de implantação de um ASIC (*Application-Specific Integrated Circuit*) personalizado possui um longo ciclo de desenvolvimento. Além disso, protocolos customizados também podem consumir um espaço precioso do ASIC, o que torna necessário uma proposição de soluções capazes de lidar com esse desafio.

Em nossa arquitetura, cabe ao *switch* de borda a tarefa de codificar os identificadores da rota principal e emergencial em todos os pacotes que serão transportados nos *switches* de núcleo. Assim, os pacotes podem ser transportados com resiliência em uma malha (*fabric*), sem que seja necessário qualquer modificação das máquina virtuais dos inquilinos no *Datacenter*. Vamos utilizar a Figura 3.2, continuando com o mesmo exemplo, onde *src* é um VM qualquer que está conectada a um *switch* de borda ($host_1$) e deseja se comunicar

com outra VM *dst* ($host_{56}$) que está conectada em outro *switch* de borda, ambas VMs em verde.

Para simplificar a explicação da implementação, assumimos que tais VMs pertencem ao mesmo inquilino e que toda comunicação já utiliza algum mecanismo de isolamento lógico, logo, o processo de comunicação segue os passos descritos a seguir: i) a VM *src* gera uma mensagem ARP solicitando o MAC de destino da VM *dst* (*arp_request*); ii) o *switch* de borda cujo o *src* está conectado irá recebe a mensagem de ARP e redireciona e mensagem de ARP para o controlador; iii) o Controlador RDNA responde a mensagem de ARP (*arp_reply*) para o *switch* de borda que redireciona para a VM *src* a resposta do ARP. Além disso, o Controlador RDNA também entrega aos dois *switches* de borda envolvidos na comunicação o IRP e o IRE, que serão utilizados no processo de encaminhamento dos pacotes pelos *switches* de núcleo. Basicamente, os identificadores são entregues utilizando uma representação de estado nos *switches* de borda, assim cada pacote terá o campo do MAC de destino reescrito com o identificador de rota principal e o campo do MAC de origem reescrito com o identificador de rota emergencial. No *switch* de borda do *dst* os endereços de MAC de origem e destino retornam ao estado original, sendo entregues pelo *switch* de borda de forma transparente para a VM do *dst*.

Optou-se em utilizar os campos do MAC de destino e de origem na implementação da arquitetura por dois motivos. Primeiro o endereço MAC ocupa uma posição privilegiada no cabeçalho Ethernet (a partir do 9° Byte do quatro Ethernet), o que permite implementação da proposta em Camada 2 [Jouet et al. 2015] com maior agilidade para leitura dos IDs e consequentemente menor latência para a comutação do pacote. A segunda vantagem em se utilizar os endereços MAC (*dst* e *src*) está na separação semântica dos identificadores IRP e IRE, ou seja, em nossa arquitetura optamos em não separar os IDs por comprimento. Está tomada de decisão também permitiu reduzir o tempo de latência na computação dos pacotes no núcleo, pois os Elementos de Núcleo não precisam efetuar nenhuma reescrita (*shift*) nos campos de cabeçalho dos pacotes, o que permite simplificar o mecanismo de encaminhamento. Diferente das abordagens utilizadas em [Rosen et al. 2001, Motiwala et al. 2008, Nguyen et al. 2011, Ramos 2013, Hari et al. 2015] que requerem a manipulação do pacote em cada salto nos *switches* de núcleo. Nossa arquitetura só modifica o pacote no *switch* que detectou a falha, se ela ocorrer. No restante dos *switches* de núcleo o pacote continuará inalterado.

Nossa análise considera o tempo de recuperação como o tempo que a rede demora para se tornar operante novamente, após a detecção de uma falha. Assumimos que a rede possui um método/protocolo de detecção de falha, por exemplo via protocolo LLDP ou Ethernet OAM com mensagens de verificação CCMs (*Continuity Check Message from Carrier Ethernet standard*) [Forum 2013]. Portanto, esse aspecto está fora do escopo de discussão nesta tese.

Na Figura 5.2 ilustramos um exemplo de representação dos estados nos *switches* de

borda. No campo do MAC de destino (*Destination MAC Address*) o identificador de rota principal (IRP), cujo valor é 86446. Da mesma forma que o identificador de rota emergencial (IRE) é inserido no campo do MAC de origem (*Source MAC Address*), cujo o valor é 117555379, todos convertidos para valores em hexadecimal.

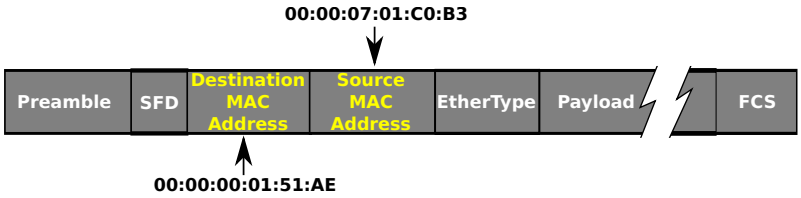


Figura 5.2: Representação dos identificadores de rota mapeado no quadro Ethernet 802.3.

Outra vantagem em utilizar o *switch* de borda está na simplicidade e eficiência em prover o balanceamento de carga e engenharia de tráfego. Essas tarefas podem ser realizadas através das informações de estado, entregues pelo controlador RDNA. Por exemplo, para realizar o balanceamento de carga, a seleção de um caminho entre os nós de comunicação de origem e destino pode ser realizado a partir de uma distribuição aleatória [Yuan et al. 2007] sobre todos os múltiplos caminhos disponíveis na topologia, todavia, sem sobrecarregar o *switch* de borda, dado que todo cálculo é realizado pelo controlador RDNA.

O controlador RDNA instala estados por fluxo em *switches* habilitados para OpenFlow. Este controle de granularidade fina permite que os *switches* de borda façam o mapeamento de campos específicos no cabeçalho dos pacotes em IDs de rota. Lembrando que o armazenamento de estado em nossa arquitetura somente acontece nos *switches* de borda. Assim, antes dos pacotes entrarem no Elemento de Núcleo, os *switches* de borda (ingresso) inspecionam e, com base nas regras da tabela de fluxo, incorporam IDs de rota no cabeçalho dos pacotes (por exemplo, utilizando um ação do OpenFlow para reescrever a partir de um campo de endereços MACs os IDs das rotas). Os identificadores de rotas são posteriormente removidos do cabeçalho dos pacotes nos *switches* de borda (egresso), reescrevendo os IDs das rotas para os respectivos endereços MACs originais).

Para os fluxos *unicast* o Controlador RDNA atribui 02 (dois) estados no *switch* de borda de ingresso e 02 (dois) estados no *switch* de borda de egresso. Esses estados são necessários para inserir e remover os respectivos IDs, mantendo a comunicação transparente para as VMs e *Containers*.

5.1.3 *Switches* de núcleo

Para o elemento de núcleo, consideramos duas diferentes implementações:

- Open vSwitch [Pfaff et al. 2009] (OvS) na versão 2.5.2 modificado;

- Placa Lógica Programável (FPGA) usando a plataforma NetFPGA SUME [1].

Vale a pena repetir que a arquitetura RDNA não exige que os elementos de núcleo sejam *switches* habilitados para SDN. A operação do módulo (mecanismo de encaminhamento) e a substituição do IRP pelo IRE (reação rápida à falha) são de fato as principais funções a serem suportadas por este elemento. No entanto, as SDNs permitem que os *switches* de núcleo notifiquem falhas secundárias ao controlador RDNA, por exemplo, se uma segunda falha ocorrer na rota emergencial. Assim, o uso de um controlador SDN fornece mecanismo de recuperação dinâmica dos IDs IRP e IRE.

Desta mesma forma, nossa arquitetura não exige que o protocolo de comunicação entre os planos de controle e dados seja o OpenFlow. Contudo, como prova de princípios adotamos o controlador Ryu e o OvS, também o OpenFlow como protocolo de comunicação entre os dos planos.

Para o elemento de núcleo consideramos duas diferentes abordagens na validação do mecanismo rápido de reação à falha, conforme apresentado no Capítulo 3. Para o ambiente emulado, consideramos a cópia do IRE substituindo o IRP, já para o ambiente utilizando as NetFPGA SUME adotamos a indicação através de um campo específico no pacote. Maiores detalhes serão apresentados na explicação de cada abordagem.

5.2 RDNA sobre a Plataforma Emulada

Utilizamos a plataforma de emulação Mininet [Lantz et al. 2010], para implementar o primeiro protótipo experimental. Portanto, quando um pacote chega ao *switch* de borda de ingresso, o pacote é enviado para o controlador RDNA que seleciona as rotas principal e emergencial entre todos os caminhos pré-calculados (origem e destino). Em seguida, o controlador envia mensagens (*flow-mod*) para instalar as regras necessárias nos *switches* de borda, que incluem o ingresso e egresso.

Direção do Fluxo	Switches de Borda	Match	Ação
Fluxo de ida	Ingresso	MAC_DST: 00:00:00:00:00:02 and MAC_SRC: 00:00:00:00:00:01	SetField(eth_dst=00:00:00:00:00:D2, eth_src=00:00:00:00:06:B4), output = 4
	Egresso	MAC_SRC: 00:00:00:00:06:B4 and IP_DST: 10.0.0.2	SetField(eth_dst=00:00:00:00:00:01, eth_src=00:00:00:00:00:02), output = 1
Fluxo de volta	Ingresso	MAC_DST: 00:00:00:00:00:01 and MAC_SRC: 00:00:00:00:00:02	SetField(eth_dst=00:00:00:00:01:6F, eth_src=00:00:00:00:06:28), output = 4
	Egresso	MAC_SRC: 00:00:00:00:06:28 and IP_DST: 10.0.0.1	SetField(eth_dst= 00:00:00:00:00:02, eth_src= 00:00:00:00:00:01), output = 5

Tabela 5.1: Exemplo de entrada nas tabelas de fluxo dos *switches* de borda.

A Tabela 5.1 detalha, por exemplo, as regras de fluxo instaladas nos *switches* de borda com as ações correspondentes para definir um fluxo entre as VMs localizadas no $host_1$

[1] A implementação foi realizada em conjunto com o Laboratório *High Performance Networks Group* (HPNG), da Universidade de Bristol, Bristol, Reino Unido - UK.

(*src*) e no $host_{56}$ (*dst*), conforme ilustrado anteriormente na Figura 3.2. No *switch* de ingresso, a regra OpenFlow inclui uma ação para adicionar os IDs das rotas no cabeçalho dos pacotes e uma ação para encaminhá-los para a rede de núcleo. Observe que, em nosso protótipo, os campos de endereço MAC foram usados para codificar os IDs IRP e IRE no cabeçalho dos pacotes. Assim, da mesma forma, no *switch* de borda de egresso a regra inclui uma ação para reescrever os endereços MAC originais e uma ação para encaminhar os pacotes para para a VM *dst*. As entradas de fluxo nos *switches* de borda são baseadas no endereço IP de destino (mas VLAN opcional ou qualquer outro identificador de inquilino desejado pode ser suportado).

Em termos de modificações do OvS, apresentamos o pseudo-código (Algoritmo 3) retirado da função *xlate_ff_group*. Aproveitamos a estrutura OpenFlow 1.3 Fast Failover [Oostenbrink et al. 2017] nos *switches* do núcleo no ambiente emulado. Assim, quando o *switch* recebe um pacote, ele verifica se a porta está disponível (Linha 3), então IRP é usado no encaminhamento. No entanto, se a porta não estiver disponível, o *switch* do núcleo verificará se o IRP é diferente do IRE (Linha 6). Se for o caso, ele substitui o IRP pelo IRE (Linha 7). Esta substituição simplifica o encaminhamento para o próximo salto, dado que para o próximo *switch* de núcleo não interessa a porta de origem do pacote, ou seja, ele apenas utilizará o IRP atualizado, ignorando possíveis falhas na rede. Portanto, para os *switches* subsequentes ao longo da rota, eles continuam fazendo a operação do módulo até que os pacotes alcancem o *switch* de borda de egresso. Somente se ocorrer uma segunda falha, os pacotes serão encaminhados para o controlador RDNA (Linha 10).

Por questões de implementação, é importante destacar que o Algoritmo 3 somente é executado quando acontece uma falha, detectada pelo modo usuário do OvS (*user mode*), ou seja, ele não é executado para cada pacote que está sendo enviado pelo *switch* de núcleo. Conforme comentado, utilizamos a função já existente para provar o princípio de recuperação utilizando o identificador IRE.

Algorithm 3 Encaminhamento dos pacotes dentro dos *switches* de núcleo.

```
1:  function XLATE_RW_GROUP(STRUCT XLATE_CTX ...)(...)
2:      ⋮
3:      if bucket then                          ▷ Verifica se a porta esta disponível
4:          enviar pacote para próximo salto;
5:      else
6:          if MAC_DST != MAC_SRC then
7:              Copiar MAC_DST ← MAC_SRC
8:              Chamar novamente a Função xlate_rw_group com o novo MAC_DST
9:          else
10:             Enviar pacote para o Controlador RDNA
11:     return Null
```

5.3 RDNA sobre a Plataforma FPGA

Com o objetivo de avaliar a arquitetura proposta em termos de latência de comutação e recuperação ultra-rápida a falhas, consideramos a implementação, validação e experimentação usando NetFPGA-SUME 10Gbps.

Na Figura 5.3 apresentamos a arquitetura do *switch* RDNA com quatro portas de entrada e saída implementadas na plataforma NetFPGA SUME. Nosso protótipo de *switch* RDNA de núcleo baseado em FPGA foi implementado considerando o suporte a comutação *wormhole* (WH) [Ni and McKinley 1993], que é uma técnica de comutação de pacotes usada para reduzir o espaço do *buffer* e consequentemente reduzir a latência [Boden et al. 1995]. Na comutação WH, os pacotes que chegam à porta de entrada são encaminhados imediatamente para a fila de saída assim que ela estiver livre. Nesse cenário, o "Switch Allocator" é usado para definir as portas de saída com base na identificação de rota (ou seja, IRP ou IRE) extraída dos pacotes por analisadores nas portas de entrada. Observe que os blocos MAC Ethernet e PCS/PMA de 10Gbps (Xilinx IP cores) não estão representados na Figura 5.3.

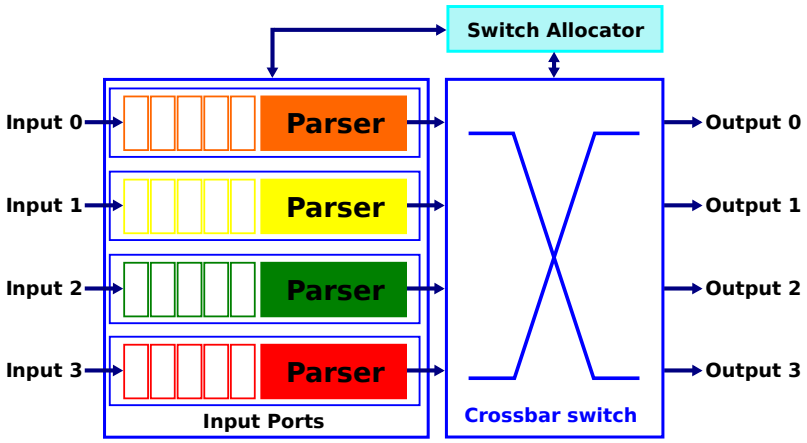


Figura 5.3: Arquitetura do Switch de núcleo utilizando plataforma NetFPGA SUME 10Gbps.

A configuração experimental utilizada em nosso *testbed* é ilustrada na Figura 5.4. Este cenário compreende 04 (quatro) placas NetFPGA SUME que representam os *switches* de núcleo na arquitetura RDNA. Cada placa suporta 04 (quatro) portas ópticas SFP+ (transmissor/receptor) com uma taxa de vazão a 10Gbps (*line rate*). Também foram configuradas na inicialização (*bootstrap*) os identificadores dos *switches*, setados para S_5, S_7, S_{11}, S_{13}. Além disso, cada placa está configurada para executar as etapas descritas no Capítulo 3, que incluem os mecanismos de encaminhamento baseados na operação de módulo, através dos identificadores (IRP e IRE) codificados nos pacotes, e o mecanismo de reação rápida à falha.

Ainda nesta mesma Figura 5.4, utilizamos dois analisadores de tráfego Anritsu diferentes (MD1230B, MT1100A) para gerar e monitorar o tráfego. Os analisadores de tráfego

considerados executam as operações do *switch* de borda (ou seja, ingresso e egresso) no pacote com os identificadores de rota (IRP e IRE).

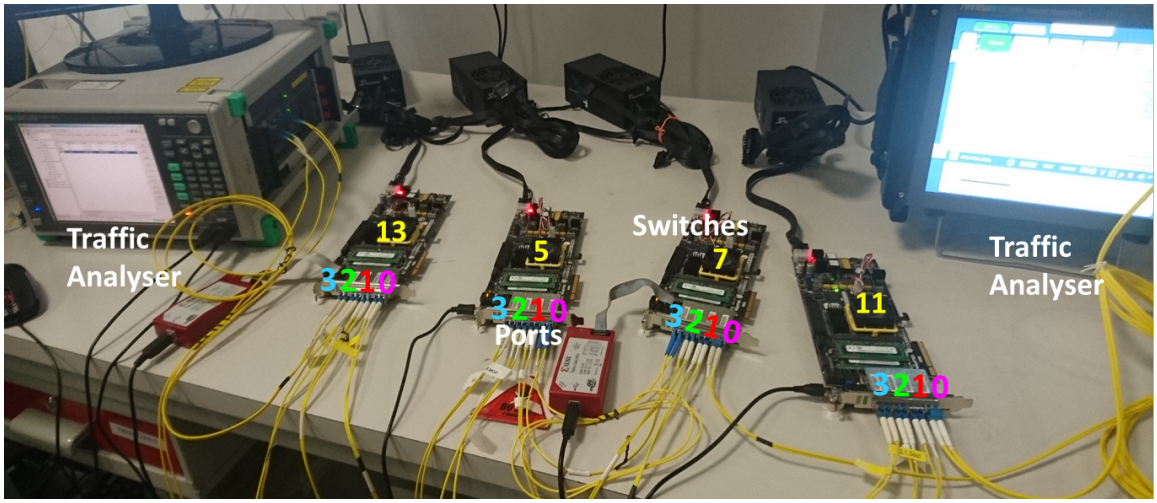


Figura 5.4: Configuração topológica do *testbed* com os *switches* de núcleo NetFPGA SUME 10Gbps, além dos geradores de tráfego utilizados.

Na Figura 5.5 apresentamos o formato de cabeçalho em alto nível estendido usado na configuração experimental. Em vez de usar os campos de endereços MAC para codificar o IRP e IRE, os IDs de rotas foram codificados no 31° (trigésimo primeiro) Byte para experimentação da arquitetura RDNA, assim, optamos em utilizar um protocolo agnóstico para fins de experimentação. Decidiu-se não utilizar os endereços MAC de destino e origem por questões de configuração dos analisadores de tráfego [2], todavia, como o objetivo era avaliar o mecanismo de encaminhamento e de reação rápida à falha, o uso de outros campos do cabeçalho não prejudicou esta análise, assim como não prejudicou a demonstração de viabilidade da arquitetura RDNA.

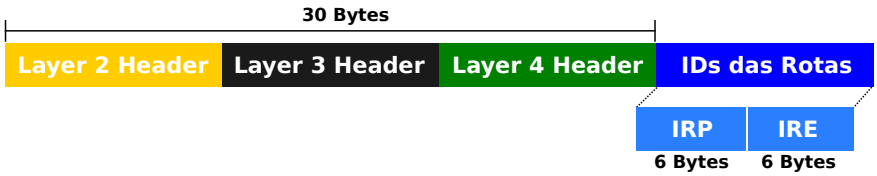


Figura 5.5: Formato do Cabeçalho para os experimentos com as FPGAs.

5.4 Validação

Esta seção é dedicada aos resultados experimentais utilizando os nossos protótipos, implementados tanto no ambiente emulado quanto na plataforma NetFPGA. Optou-se em dividir a avaliação em quatro partes, descrito nas próximas subseções. Os principais objetivos da avaliação incluem: i) avaliar o impacto da reação rápida à falha na vazão do

[2]Não foi necessário alterar a configuração padrão dos equipamentos utilizados.

protocolo TCP, ii) realizar uma comparação do tempo de recuperação de falhas, iii) medir a latência da RDNA na plataforma NetFPGA com reação rápida à falha, e finalmente, iv) comparamos a latência de comutação de pacotes entre a RDNA versus SDN OpenFlow tradicional.

5.4.1 Impacto da reação rápida à falha na vazão do TCP

Como nosso foco está em prover uma infraestrutura eficiente para atender a demanda por conectividade dentro dos *Data Centers*, concentramos nossos esforços em demonstrar como a RDNA pode beneficiar essas aplicações. No primeiro cenário ilustramos a troca de dados entre dois serviços que estão rodando em diferentes *hosts* físicos. Este cenário é comumente encontrado em DCs, portanto, em caso de falha a rede deve oferecer um mecanismo rápido de recuperação.

Neste cenário avaliamos a eficiência do processo de recuperação de falha nos enlaces. Selecionamos diferentes técnicas de recuperação, incluindo mecanismos de proteção reativa/proativa. O objetivo foi avaliar a granularidade do tempo de recuperação de falhas e seu impacto na taxa de transferência (*throughput*) do TCP. Optou-se em utilizar o TCP pela relação de controle do protocolo, desta forma é possível observar o pior caso na recuperação à falha, ou seja, que exige o retorno da confirmação de recebimento. Os experimentos foram realizados no ambiente emulado onde a topologia de rede ilustrada na Figura 3.2 foi implementada. Durante os experimentos, os enlaces são desconectados usando o comando Linux *ifdown*. Os mecanismos de recuperação de falhas considerados para esta análise são os seguintes:

- Reativo: o controlador aguarda uma notificação de falha do enlace do *switch* de núcleo. Após a notificação, atualiza as regras da tabela de fluxo para todos os *switches* de núcleo que pertencem a rota selecionada. Este é o caso *baseline* de restauração de falha (pior caso).

- OpenFlow 1.3 Fast Failover [Adrichem et al. 2014] (FF): o controlador instala entradas FF nos *switches* S_{11}, S_{19} e S_{23}. No caso de falha, uma nova entrada deve ser adicionada nos *switches* S_{29} e S_{13} pelo controlador RDNA.

- Totalmente proativo: as regras FF são instaladas e configuradas em todos os *switches* para ter uma rota de *backup* para proteger o fluxo de uma falha em qualquer enlace de toda a rota. Esse é o melhor cenário em termos de tempo de recuperação de falhas.

- RDNA: os fluxos são instalados apenas nos *switches* de borda. Os *switches* de núcleo reagem a falha no enlace apenas substituindo o IRP pelo IRE codificado no pacote. As portas de saída são então calculadas de acordo com a operação de

módulo da rota emergencial (IRE), orientando o fluxo através da rota emergencial até o destinatário.

Na Figura 5.6 apresentamos os resultados de vazão TCP (taxa limitada em 1 Gbps) para a falha no enlace $S_{37} \rightarrow S_{53}$ (conforme Figura 3.2) para os diferentes mecanismos de recuperação de falhas avaliados. Conforme esperado, o pior cenário é a recuperação reativa, pois depende da comunicação com o plano de controle reduzindo a vazão do TCP consideravelmente. Todavia, apesar da redução no tempo de recuperação de falhas pelo OF Fast Failover, a vazão do TCP é drasticamente afetada porque a rota não foi totalmente protegida. Além disso, embora o Failover seja interessante, ele é um rápido mecanismo de proteção apenas local, ou seja, há um trabalho adicional para os administradores que devem adicionar entradas específicas em cada *switch* ao longo da rota, tanto para os fluxos de ida quanto para os fluxos de volta (preventivamente).

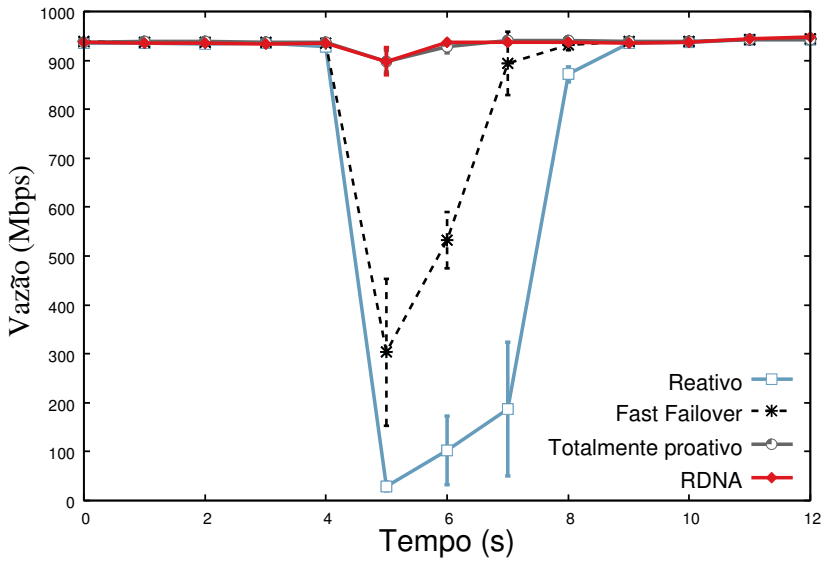


Figura 5.6: Análise de vazão do TCP em Mbps para o tempo de recuperação com diferentes técnicas e intervalo de confiança de 95%.

Ainda na Figura 5.6, observe que o mecanismo Totalmente proativo mantém a vazão média do TCP em 897 Mbps (±27). Entretanto, este mecanismo exige estados adicionais de proteção em todos os *switches* de núcleo ao longo da rota. Em contraste, a arquitetura RDNA forneceu basicamente a mesma vazão do mecanismo de proteção Totalmente proativo, porém com menos complexidade, uma vez que não há necessidade de adicionar nenhuma entrada de fluxo nos *switches* de núcleo. Além disso, este resultado mostrou que as ações de reescrita de cabeçalho dos pacotes realizada pelos *switches* de borda (inserir e remover IRP e IRE) e o desencadeamento do roteamento emergencial no *switch* que detectou a falha de enlace não prejudica o desempenho em termos de vazão na rede.

5.4.2 Tempo de recuperação da falha

Neste segundo cenário avaliamos especificamente o mecanismo totalmente proativo contra a proteção do RDNA. O objetivo é portanto comparar o tempo de recuperação de falhas, demonstrando os limites fornecidos pela arquitetura RDNA para as aplicações e serviços que serão executados dentro do DC.

Para medir com precisão de microssegundos, usamos uma ferramenta chamada Metherxis [Mafioletti et al. 2016]. Esta ferramenta permite a geração de uma carga de trabalho constante simulando a transmissão de um serviço com troca de dados dentro da rede do DC. Com a finalidade de medir o tempo médio de recuperação, utilizamos a ferramenta enviando à uma taxa de 1000Mbps durante 30s (trinta segundos) entre VMs *src*, localizada no $host_1$, para o *dst* $host_{56}$ por 30 (trinta) rodadas. Os *timestamp* dos pacotes foram armazenados em arquivo para análise posterior. Uma falha de enlace é injetada entre os *switches* S_{37} e S_{53}, utilizando o comando *ifdown*.

Na Figura 5.7 apresentamos o tempo de recuperação para o mecanismo totalmente proativo versus a proteção RDNA no ambiente emulado Mininet. Como pode ser visto, embora haja uma variabilidade maior no totalmente proativo, quando comparamos com o RDNA, ambas as técnicas são capazes de fornecer garantias semelhantes de tempo de recuperação (sub-milissegundos, em torno de 550 μs). Portanto, elas podem ser considerados equivalentes em termos estatísticos. Isto ocorre porque as duas utilizam o mesmo mecanismo de detecção de falha (*Fast Failover*), dado que não consideramos a implementação de outro mecanismo de detecção de falhas na rede para provar nosso princípio utilizando o identificador emergencial (IRE).

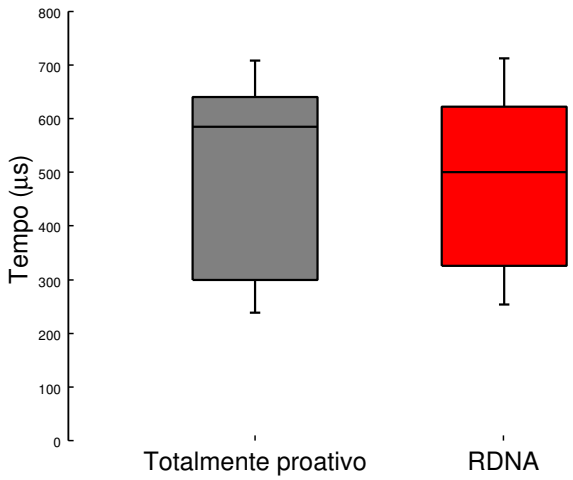


Figura 5.7: Comparação do tempo de recuperação Totalmente proativo versus RDNA.

No entanto, nossa arquitetura não depende da tabela de fluxos, logo, não sofre variação com a taxa de ocupação da memória. Outra desvantagem da abordagem Totalmente proativo, é que para garantir a proteção de fluxo usando o OF, é necessário instalar 04 (quatro) regras do OpenFlow em cada *switch* de núcleo (02 (duas) regras para a ida e 02 (duas)

regras para a volta). Pegando como exemplo o *switch* HP ProVision 10Gbps Ethernet [Stephens et al. 2012, Agarwal et al. 2014], que suporta 1500 fluxos na TCAM, apenas 375 fluxos podem ser totalmente protegidos. Além disso, conforme comentamos as regras de fluxo armazenadas fora da TCAM, ou seja, em memórias mais lentas (SRAM), são significativamente impactadas sofrendo com a variabilidade de latência [Long et al. 2012]. Por outro lado, o RDNA reduz o estado de encaminhamento dos pacotes codificando a informação da rota dentro do próprio pacote sem gerar sobrecarga ou encapsulamento, pois limita o estado dos micro-fluxos aos *switches* de borda, eliminando a necessidade de manter qualquer estado de micro-fluxo nos *switches* do núcleo. Portanto, além de oferecer um mecanismo eficiente de reação rápida à falha dentro do núcleo da topologia, a RDNA também fornece a escalabilidade necessária para atender a demanda por conectividade, onde não requer nenhuma tabela de estado no núcleo da rede.

5.4.3 Baixa latência e reação rápida à falha

O ambiente emulado nos permitiu avaliar a arquitetura RDNA considerando diferentes técnicas de recuperação de falha. Contudo, devido a limitação inerente do próprio ambiente, projetamos e implementamos um segundo protótipo. Nosso objetivo neste novo ambiente é melhorar a precisão das medidas de latência dentro do DC, inclusive considerando situações de falha, deste modo, projetistas e desenvolvedores de serviços e aplicativos podem utilizar estes resultados como parâmetro linear na arquitetura RDNA.

Esta segunda versão do protótipo foi implementado na plataforma NetFPGA. Cada NetFPGA é composta por 04 (quatro) interfaces Ethernet de 10G (veja Figura 5.4) e o cenário de validação utilizado é uma topologia de malha completa como mostrado na Figura 5.8. Devido à limitação do número de placas FPGA (04), apenas os *switches* de núcleo foram considerados em nosso *testbed* com e sem falha de conexão entre os enlaces.

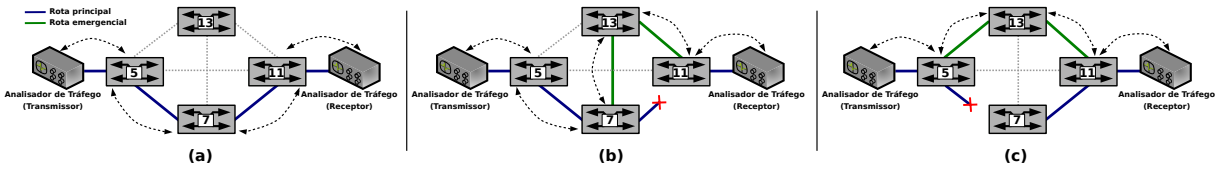
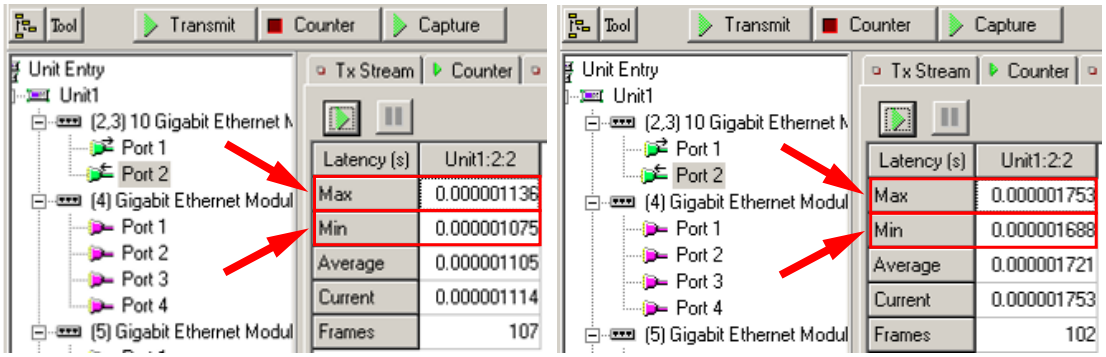


Figura 5.8: Topologia Full mesh com 04 (quatro) *switches* de núcleo: (a) sem falha (3 saltos); (b) falha no enlace entre S_5 e S_7 (3 saltos); e (c) falha no enlace entre S_7 e S_{11} (4 saltos).

A Figura 5.9 apresenta a captura de telas do analisador de tráfego que exibem as medidas de latência para os experimentos com 01 (um) e 02 (dois) saltos. Na Figura 5.10 complementamos as medidas de latência mostrando os valores de latência para 03 (três) saltos. Para se obter a contribuição exata de latência em cada salto (acréscimo na comutação), devemos subtrair os 0.5 μs, que foram medidos na conexão de porta *back-to-back* (B2B) do analisador de tráfego (com comprimento do pacote programável de 1500 Bytes)

do total acumulado de latência. Assim, a latência média para um *switch* RDNA de núcleo é em torno de 0.6 μs. Este valor médio foi obtido com base na coleta de 30 (trinta) amostras com duração de 30s (trinta segundos) por amostra.



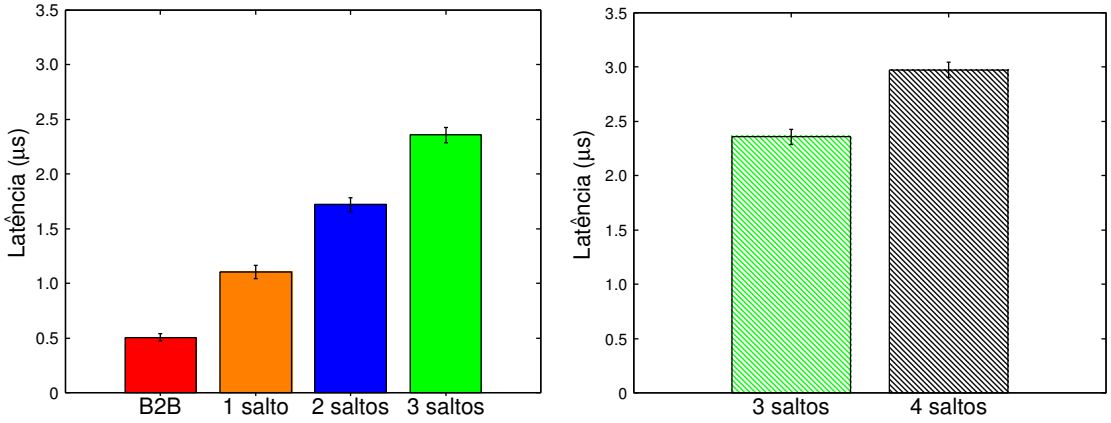
(5.9.1) *01 (um) salto.* (5.9.2) *02 (dois) saltos.*

Figura 5.9: *Screenshot* das telas no analisador de tráfego com as medidas de latência do *switch* de núcleo (sem falha).

Uma importante observação obtida através deste experimento foi que a variabilidade de latência (*jitter*) está em ordem de dezenas de nanosegundos, ou seja, pode ser desconsiderada (*no jitter*). Portanto, a RDNA se apresenta como uma arquitetura promissora na comutação de pacotes, capaz de garantir latência de encaminhamento determinística na comunicação *host* a *host*, mesmo em período de falhas da rede, isso se deve ao fato principalmente do modelo de recuperação utilizado em nossa arquitetura. No *switch* de núcleo, para que o processo de reação rápida à falha aconteça, basta utilizar as informações do identificador de rota emergencial que já foi programado para contornar a falha por um caminho tipicamente disjunto ao caminho principal. Para isso é sabido que um mecanismo de detecção de falha monitore o enlace/porta, notificando apenas o *switch* local de uma possível alteração.

Para os casos de falhas de enlace representadas nas Figuras 5.8 (b) (c), existem dois casos diferentes com 4 e 3 saltos, respectivamente. Observe que não há mecanismo de detecção de falhas de enlace implementado internamente na FPGA. Seria necessário usar componentes adicionais (por exemplo, medidores de energia) para detectar uma possível falha no enlace. Em vez disso, decidimos emular as falhas utilizando um campo especial dentro do pacote programável. Este foi adicionado após o IRP e IRE (anteriormente mostrado na Figura 5.5), este campo adicional possue o comprimento de 1 Byte é usado para carregar o ID do *switch* que irá emular a falha. Ao receber o pacote, o *switch* cujo ID corresponda à identificação do *switch* codificado dentro do pacote (por exemplo, S_7 como no exemplo mostrado na Figura 5.8 (b)) desencadeia a configuração da rota emergencial, utilizando o IRE já codificado no pacote. Nesse caso, o *switch* onde a falha do enlace é emulada introduz um atraso de 25.6ns (4 ciclos de *clock*) para executar este processo.

Na Figura 5.10.2 ilustramos o quão rápido é a nossa reação à falha, pois consiste apenas em utilizar o IRE sem depender de nenhum protocolo de convergência ou resposta de uma entidade externa (controlador de rede). A latência medida foi em torno de 2.93μs para 04 (quatro) saltos e 2.33μs para 03 (três) saltos. Uma observação importante é que, para a latência medida dentro do núcleo RDNA, não consideramos a contribuição do tempo de detecção de falhas, portanto, a latência medida é claramente um tempo de reação à falha pura. Entretanto, observe que a arquitetura RDNA fornece um mecanismo ultra-rápido de reação à falha, evidenciado em nosso segundo protótipo.



(5.10.1) *Sem falha.* (5.10.2) *Com falha.*

Figura 5.10: Medida de latência dos *switches* de núcleo.

5.4.4 Desempenho no encaminhamento de pacotes (RDNA versus SDN OpenFlow tradicional)

As abordagens tradicionais de SDN OpenFlow têm se apresentado como propostas inovadoras. Entretanto a arquitetura SDN OpenFlow tradicional implica que todo *switch*, em qualquer posição da rede, processa pacotes unicamente com operação de pesquisa na tabela (*lookup table*), baseando-se em centenas de *bits* presentes nas múltiplas tabelas, o que pode levar a uma significativa variabilidade, além de alta latência de encaminhamento dos pacotes.

Para entender as limitações da arquitetura SDN OpenFlow tradicional, considere a Figura 5.11 **(a)**. Observe que o *pipeline* de processamento do pacote nos *switches* SDN OpenFlow possuiu a seguinte sequência: i) enfileiramento na porta física de entrada; ii) análise e pesquisa das regras de fluxo nas tabelas; iii) execução do conjunto de ações correspondentes à entrada do pacote no *switch*; iv) dependendo da configuração desejada nos *switches* OpenFlow, uma nova correspondência e pesquisa nas tabelas de fluxo podem ser realizadas, incluindo a execução de outras ações que correspondem ao processamento de saída do pacote; por último v) o pacote é enfileirado na porta física de saída [ONF 2012].

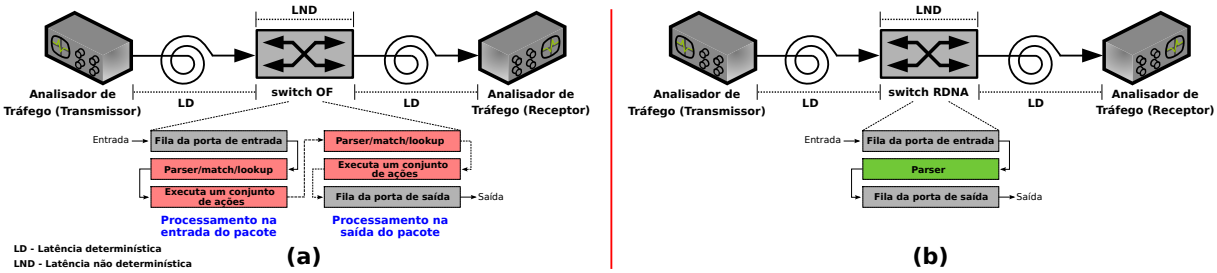


Figura 5.11: Fluxo dos pacotes através do *pipeline* de processamento: **(a)** visão genérica do encaminhamento dos pacotes utilizando SDN via OpenFlow tradicional [ONF 2012]; **(b)** encaminhamento dos pacotes nos *switches* de núcleo RDNA.

Ainda na Figura 5.11 **(a)**, note que ao receber um pacote o *switch* SDN OpenFlow direciona o mesmo para uma fila na respectiva porta de entrada; em seguida, ele executa a análise do pacote, procurando nas tabelas a correspondência entre os campos do cabeçalho configurados nas regras de fluxo. Neste ponto, observe que devido as características do protocolo OpenFlow, a pesquisa pode ser realizada em múltiplas tabelas suportadas inclusive em *hardware*, como as TCAMs. Todavia, mesmo memórias especiais como as TCAMs enfrentam consideráveis desafios para garantir latência determinística de busca na tabela. Isto ocorre devido suas características, por exemplo, a TCAM é compartilhada entre todas as portas de entrada, portanto, o tempo de busca e sua variação vai eventualmente depender da taxa de chegada dos pacotes nas interfaces e da quantidade de portas de entrada, além do respectivo tempo de enfileiramento para a realização da busca [Congdon et al. 2014]. Outro ponto importante, que limita esse tipo de memória, é o mecanismo de pesquisa (*lookup*), que permite apenas uma busca por ciclo de *clock* [Yamanaka et al. 2014], independentemente da quantidade de estados armazenados em sua memória, logo, inevitavelmente haverá fila para acessar a TCAM, conforme discutido em [Congdon et al. 2014]. Ademais, os *switches* OpenFlow tradicionais consideram em seu *pipeline* a atualização de contadores estatísticos, campos do cabeçalho, além do próprio pipeline de processamento. Todas essas ações incrementam a latência de encaminhamento dos pacotes dentro desses *switches*.

Ao contrário, conforme apresentamos na Figura 5.11 **(b)**, o *pipeline* dos *switches* de núcleo RDNA é mais simples e apesar de não garantir latência de encaminhamento determinística, devido as filas de entrada e saída, nosso modelo de encaminhamento fornece o mesmo tempo para a operação de encaminhamento dos pacotes dentro do *switches* RDNA. Outra potencial vantagem da arquitetura RDNA é que várias operações de encaminhamento paralelas podem ser realizadas por ciclo de *clock*, além disso, nosso original mecanismo de reação à falha oferece latência de encaminhamento determinística, não dependendo de nenhuma atualização de tabela ou protocolo de convergência no processo de recuperação de falhas na rede.

Com o objetivo de comparar o desempenho no encaminhamento de pacotes entre a

arquitetura RDNA contra a abordagem tradicional SDN OpenFlow, nós avaliamos nosso *switch* de núcleo implementado na NetFPGA SUME versus um *switch* OpenFlow de mercado. A Tabela 5.2 apresenta resumidamente as especificações desses *switches*.

Tabela 5.2: Resumo das especificações técnicas dos *switches* avaliados.

Marca e modelo	Capacidade (máxima)	Portas 10G (SFP+)	Processador	Memória	Suporte OpenFlow
CORSA DP2100	100G	até 32	Intel Core	16GB DDR3, 120GB HD	1.3
NetFPGA-SUME	100G	até 4	Virtex-7 XC7V690T FPGA	4GB DDR3 SODIMM	-

Neste cenário, consideramos avaliar os *switches* seguindo as especificações contidas na RFC2544 [Bradner and McQuaid 1999], que descreve o procedimento de avaliação de dispositivos, incluindo os tamanhos para pacotes Ethernet, variando entre 64, 128, 256, 512, 1024, 1280 e 1518 Bytes, tempo das rajadas e quantidade de amostras. Do mesmo modo descrito anteriormente, utilizamos os analisadores de tráfego para gerar os fluxos entre origem e destino. Consideramos na arquitetura RDNA especificamente o *switch* de núcleo S_5 como (*device under test*). Para a arquitetura SDN OpenFlow tradicional, consideramos o *switch* CORSA DP2100 com apenas a inserção de regras de fluxos (*in_port action*), ou seja, a configuração mais simples possível para a realização do encaminhamento no *switch* OpenFlow. O gráfico da Figura 5.12 já considera o desconto do respectivo tempo de propagação do pacote no meio, neste caso, fibra óptica que é de 5ns por metro [Testa and Pavesi 2017].

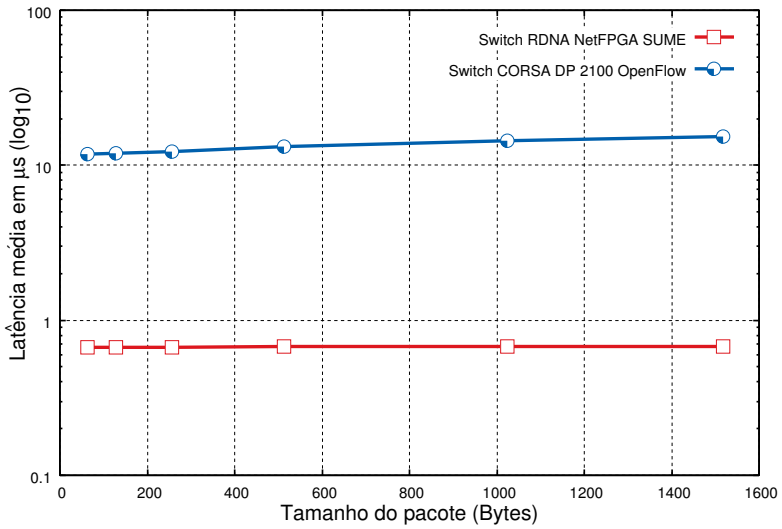


Figura 5.12: Latência de encaminhamento variando o tamanho dos pacotes. Arquitetura RDNA versus SDN OpenFlow.

Na Figura 5.12 apresentamos o resultado desta avaliação. Para todos os tamanhos de pacote na RDNA a latência de encaminhamento foi menor que 1 μ. Por sua vez, a arquitetura SDN tradicional utilizando o switch CORSA DP2100 apresentou latência média de

encaminhamento entre 11 e 15 μs, para os pacotes de 64 e 1518 Bytes, respectivamente. Este resultado demonstra que a arquitetura RDNA é mais eficiente no encaminhamento de pacotes com latência, do que a arquitetura SDN OpenFlow no *switch* CORSA DP2100, sendo uma ordem de magnitude menor.

Em relação a baixa variabilidade na latência de encaminhamento da RDNA, observada nos resultados da Figura 5.12. Creditamos está eficiência ao projeto do *switch* que foi implementado na FPGA, que é baseado no roteamento WH, conforme mostrado anteriormente na subseção 5.3. Deste modo, a RDNA evidência seu potencial no encaminhamento de pacotes no núcleo da topologia com característica similares a comutação de circuitos. Ao contrário, o *switch* CORSA utiliza um *buffer* de 6GB [3] para armazenamento do pacotes (*store-and-forward*), portanto, a variação no tamanho dos pacotes acaba influenciando na latência de encaminhamento para este modelo de *switch*, requerendo primeiro o armazenamento do pacote em seu *buffer* para posterior encaminhamento. Esta abordagem não é eficaz em termos de redução de latência para o mecanismo de encaminhamento, entretanto, é comum para todos os *switches* OpenFlow, que deve aguardar vários cabeçalhos de protocolos a fim de possibilitar a identificação do fluxo [Congdon et al. 2014].

5.5 Considerações Finais

Na (Seção 5.1), descrevemos em detalhes a implementação da arquitetura RDNA. Para demonstrar a viabilidade apresentamos o projeto e a implementação de dois diferentes protótipos: i) ambiente emulado (Seção 5.2); e ii) testbed 10 Gbps em NetFPGA SUME (Seção 5.3). A arquitetura RDNA é independente de topologia e mostrou seu potencial benefício para as Redes 2-tier Clos Network. Na Seção 5.4, apresentamos e discutimos os resultados experimentais utilizando nossos protótipos. A RDNA atinge baixa latência de comutação por salto ($\approx 0.6\mu$s) e nenhuma variação de latência no encaminhamento dentro do núcleo da rede.

Além disso, nossa arquitetura oferece proteção ao longo de toda rota principal com recuperação de falha ultra-rápida (sub-milissegundos). Este resultado é extremamente desejado para serviços e aplicações de missão crítica de forma geral. É importante destacar que em todos os cenários avaliados, consideramos apenas a comunicação dentro da infraestrutura do DC. Também não incluímos o tempo de processamento e armazenamento dos serviços e aplicações, consideramos que essas variáveis poderiam criar ruídos para a avaliação da arquitetura, e por isso foram descartados em nossos experimentos. Deste modo, optamos em demonstrar os limites inferiores da RDNA e como ela pode ser utilizada para atender a crescente demanda por conectividade na nuvem.

Ainda na Seção 5.4, comparamos dois projetos para arquitetura de *switches*, com diferentes métodos de comutação. O método de comutação de pacotes *cut-through switching*

[3] Conforme datasheet disponível no site do fabricante (https://www.corsa.com/products/dp2100/).

não requer o armazenamento completo do *frame*, portanto, após receber e processar as informações do cabeçalho necessárias para comutação, o pacote já pode ser enviado para a fila da porta de saída. Já o método *store-and-forward switching* vai requerer no mínimo o armazenamento de todo *frame* em um *buffer* temporário. Portanto, um *switch* OpenFlow sempre vai utilizar o método *store-and-forward switching*, dado sua característica arquitetural. Outro problema na utilização de *switches* OpenFlow tradicionais em DCs é o uso das TCAMs, que além de consumir muita energia seu uso, dificulta o atendimento aos serviços de baixa latência na conectividade da nuvem. [Congdon et al. 2014] demonstrou em seu estudo que mesmo com uma única interface, haverá fila para acessar a TCAM, dependendo da taxa de chegada de novos fluxos, e que com mais interfaces o tempo de comutação vai piorar.

Ao contrário a arquitetura SDN OpenFlow tradicional, a RDNA não requer nenhum estado armazenado em tabelas, o que permitirá desenvolver no futuro um dispositivo mais barato para DCNs, além de consumir menos energia. Ademais, a RDNA suporta a implementação do método de comutação *cut-through*, que reduz a latência de encaminhamento, conforme demonstramos no protótipo implementado nas NetFPGAs. Para finalizar, destacamos que ao tornar o tempo de *lookup* determinístico, abrimos uma nova linha de investigação de políticas para engenharia de tráfego que podem levar a serviços determinísticos, entretanto, esta linha de investigação não será analisada por estar fora do escopo desta tese e serão investigados nos trabalhos futuros.

Capítulo 6

Trabalhos Relacionados

O presente capítulo apresenta alguns dos trabalhos relacionados relevantes encontrados na literatura científica da área. Deixamos propositalmente este capítulo para o final da tese, desta forma o leitor já possui o completo conhecimento das inovações aqui apresentadas. Portanto, ele pode identificar nossas principais contribuições e as diferenças da nossa arquitetura com as existentes. Esses trabalhos foram divididos em três categorias, descritas resumidamente a seguir, sendo que cada uma compõe uma seção. São elas:

- **Encaminhamento de pacotes baseado em rótulos:** discutiremos alguns trabalhos encontrado na literatura que baseiam suas decisões de encaminhamento em rótulos/etiquetas.
- **Roteamento resiliente:** nesta categoria apresentaremos algumas propostas no contexto de roteamento resiliente que podem ser aplicadas em DCNs.
- **Comunicação multicast:** discutiremos nesta categoria alguns trabalhos envolvendo comunicação multicast suportadas para DCNs.

6.1 Encaminhamento de Pacotes baseado em Rótulos

Na literatura encontramos uma numerosa quantidade de propostas e arquiteturas que fazem uso do roteamento por segmento. Também é comum encontrar nesses trabalhos o uso de mecanismos de encaminhamento baseado em rótulos ou etiquetas, incluindo: MPLS [Rosen et al. 2001], VxLANs [Mahalingam et al. 2013], SlickFlow [Ramos et al. 2013], KeyFlow [Martinello et al. 2014], Segment Routing [Filsfils et al. 2014] e Path Switching [Hari et al. 2015]. Ao longo desta seção apresentaremos as principais características destes trabalhos, iniciando com o *Multiprotocol Label Switching* (MPLS).

MPLS versus RDNA

O MPLS é uma tecnologia que domina o núcleo das infraestruturas de DC pelo mundo. Com um mercado equivalente a 3.2 bilhões de dólares anuais [Hardesty 2017], um verdadeiro império que pode ser confrontado com nossa arquitetura. Sua tecnologia utiliza rótulos inseridos em cada pacote para estabelecer uma espécie de circuito virtual, que permite rotear os pacotes pela rede de núcleo, por meio do rótulo ao invés de usar o endereço IP. Têm seu padrão especificado pelo *Internet Engineering Task Force* (IETF), que descreve o *design*, roteamento e encaminhamento dos pacotes nas redes IP. O MPLS opera entre as Camadas 2 e 3, portanto, pode ser considerada como protocolo de Camada 2.5. Ele pode ser usado para transportar vários tipos diferentes de tráfego, incluindo pacotes IP, ATM, SONET e Ethernet [Rosen et al. 2001]. Resumidamente, sua arquitetura é composta por um identificador, ou rótulo, uma classe equivalente para encaminhamento, um roteador de borda (LER), um roteador de núcleo (LSR), um protocolo de distribuição de rótulos (LDP).

O rótulo no MLPS é atribuído no cabeçalho de comprimento fixo, que além do *label* de 20 bits, contêm um campo para classe de serviços com 3 bits, um campo para empilhamento com 1 bit e um campo de TTL com 8 bits, totalizando 4 Bytes de cabeçalho. A função do rótulo é identificar a classe equivalente para encaminhamento do pacote na rede. Ao utilizar as classes de equivalência o MPLS não somente simplifica o processo de comutação dos pacotes, mas também combate o crescimento das tabelas de encaminhamento nos roteadores de núcleo.

Apesar da redução dos estados nos roteadores de núcleo é importante destacar que o rótulo usado no cabeçalho do MPLS, e atribuído em cada pacote que tráfega no núcleo da rede, só possui significado local. Portanto, exige armazenamento em tabelas de encaminhamento com as classes de equivalência, além da troca dos rótulos no pacote a cada salto. Estas são duas grandes desvantagens quando comparamos a tecnologia do MPLS com a RDNA, que utiliza um identificador de rota que possui significado global nos *switches* de núcleo, além de não exigir nenhuma mudança no cabeçalho do pacote a cada salto em codições normais da rede.

O MPLS argumenta que o uso das classes de equivalência permitem simplificar a implementação da agregação de serviço na rede. Esta simplificação é benéfica à engenharia de tráfego, pois organiza a distribuição dos fluxos na rede de modo a evitar o congestionamento causado pela acomodação desigual dos fluxos nos enlaces. Todavia, estas soluções não são novas e nem exclusivas do MPLS, várias outras propostas foram desenvolvidas com o intuito de agregar qualidade de serviço à rede, tais como as arquiteturas IntServ e DiffServ em um ambiente IP clássico [Harju and Kivimaki 2000].

A engenharia de tráfego, faz com que a operação de troca de dados na rede seja eficiente e confiável enquanto há uma otimização de seu desempenho [Gimenez et al. 2006]. O

MLPS estabelece uma variedade de módulos de controle para atender a este requisito. Em nossa arquitetura isso não é diferente, mas é realizado de maneira simples e flexível. Cabe ao elemento de gerenciamento dar suporte às políticas refinadas de distribuição dos fluxos. Deste modo, diferentes critérios podem ser implementados pelo operador do DC via controlador RDNA.

No MPLS, para habilitar a engenharia de tráfego, é necessário usar o MPLS-TE (*Traffic Engineering*), que exige o uso de protocolos de distribuição de rótulos, além do uso adicional de outros protocolos como: *ReSerVation Protocol* (RSVP), Serviços Integrados (IntServ) e *Constrained* SPF (CSPF) do OSPF, o que demanda muita configuração adicional [Awduche and Jabbari 2002]. Na RDNA apenas o elemento de borda atua orientado pelo elemento de gerenciamento, o que proporciona flexibilidade nas DCNs.

O roteador de borda no MPLS, chamado de *Label Edge Router* (LER), tem como principal função conectar uma rede de domínio MPLS com um outro domínio não MPLS. Além disso, cabe a este roteador a função de atribuir ao pacote uma classe particular de equivalência. Tipicamente, a tarefa de atribuição das classes é realizada pelos operadores, que desejam ter controle total de sua rede, dispensando o uso de protocolos dinâmicos. No entanto, essa decisão representa um desafio para as DCNs, principalmente pela carência para encontrar profissionais habilitados para configuração e manutenção do MPLS (OPEX) [Ionescu-Graff et al. 2004]. Em nossa arquitetura funcionalidade similar é realizada pelo elemento de borda, que tem o objetivo de atribuir os identificadores de rota via controlador RDNA, dando ao operador total controle a partir de uma visão centralizada da rede, o que simplifica consideravelmente o gerenciamento das DCNs.

O *Label Switching Router* (LSR) é o roteador de núcleo na arquitetura MPLS. Sua função é associada ao encaminhamento do pacote utilizando o rótulo. Basicamente, ele extrai do cabeçalho MPLS o *label* que é usado para pesquisar na tabela de encaminhamento, identificando a porta de saída e o novo rótulo que será codificado no pacote antes de encaminhá-lo para um dos seus vizinhos. O MPLS tem habilidade de usar o roteamento IP para anunciar e estabelecer comunicação na topologia, com a vantagem de utilizar o rótulo para a comutação dos pacotes. Isto representa uma vantagem que permite reduzir, mas não elimina, o armazenamento dos estados nas tabelas de encaminhamento.

Nossa arquitetura, ao contrário, elimina completamente qualquer necessidade de armazenamento de estados nos *switches* de núcleo. Outra vantagem no uso da RDNA é rapidez na recuperação à falha. A RDNA utiliza um identificador emergencial único que permite proteger toda a rota principal sem que seja necessário nenhuma troca de mensagem entre os *switches* de núcleo. O MPLS requer o armazenamento adicional de estados em cada LSR como mecanismo de proteção ao longo de toda rota principal. Além disso, requer o suporte de um protocolo de sinalização, como o *Label Distribution Protocol* (LDP) para os comutadores MPLS Fast Reroute [Swallow et al. 2005] habilitados.

Por último mas não menos importante está o estabelecimento do caminho. Nas redes

MPLS a transmissão de dados ocorre em caminhos associados a um conjunto de rótulos denominado LSP. Os *Label Switching Paths* (LSPs) são uma sequência de rótulos atribuído a cada um dos roteadores de núcleo ao longo do caminho entre a origem e o destino. Os LSPs são estabelecidos via LDP antes da transmissão dos dados (proativamente) ou após a detecção de um novo fluxo de dados (reativamente) [Gimenez et al. 2006]. Em nossa arquitetura o caminho ou rota é definido no elemento de gerenciamento (Plano de Controle), de forma proativa ou reativa. Todavia, não precisamos sinalizar ou atribuir nenhuma informação adicional aos *switches* de núcleo, basta saber apenas os identificadores internos dos *switches* e as respectivas portas de saída para estabelecimento do caminho. Todas essas informações são coletadas via elemento de gerenciamento na inicialização da topologia e são suficientes para determinar a rota entre uma origem e um destino. Coletas temporais podem ser realizadas via módulos de TE (*Traffic Engineering*) que podem ser instanciados no elemento de gerenciamento, contudo, nesta proposta não consideramos os módulos de TE.

Uma limitação da nossa proposta está no uso de um elemento centralizado para definição das rotas principais e emergenciais. O MPLS permite a implementação em conjunto com outros algoritmos de roteamento, como OSPF, para distribuição dos rótulos, o que de certa maneira torna as redes MPLS mais autônomas quando comparadas com a nossa abordagem. Todavia, o uso de elementos de gerenciamento logicamente centralizado e fisicamente distribuídos podem ser aplicados em nossa arquitetura, isso permite incluir redundância não apenas do serviço disponibilizado por este elemento, mas também resiliente para os estados configurados nos elementos de borda utilizado em nossa arquitetura. De forma colaborativa exploramos a resiliência no plano de controle e maiores detalhes podem ser encontrado em [Spalla et al. 2014, Spalla et al. 2015**?**].

VLAN e extensões versus RDNA

Uma forma simples de reduzir os estados nas tabelas de encaminhamento, é utilizar um meio termo entre uma rede de Camada 2 e Camada 3. As VLANs permitem a criação de múltiplos domínios de *broadcast* em uma camada lógica de nível 2. Ao contrário de uma LAN física, uma VLAN pode ser logicamente definida, independentemente da localização dos *hosts* na rede. Ao dividir uma grande rede em várias VLANs de dimensões adequadas, os administradores podem reduzir a sobrecarga de transmissão de *broadcast* em cada uma delas, e também garantir o isolamento entre os diferentes grupos. Comparado com o IP, VLANs simplificam a mobilidade, pois os *hosts* mantém seus endereços IP enquanto circulam em uma mesma VLAN [Ramos 2013]. Embora viável para redes de menor escala, VLANs também sofrem de uma série de inconvenientes. Elas devem ser configuradas em cada *switch*, e exigem que os recursos de largura de banda sejam explicitamente atribuídos a cada VLAN em cada *switch* participante, limitando a flexibilidade para alterar dina-

micamente os padrões de comunicação. Outro desafio em utilizar VLANs em DCs está na limitação imposta pelo campo do cabeçalho de 12 bits para os identificadores (VLAN IDs), o que limita em até 4096 diferentes túneis de isolamento para as redes virtuais.

Para resolver os desafio relacionados a VLAN surgiram as *Virtual Extensible* LAN (VxLANs) [Mahalingam et al. 2013]. Resumidamente, uma VxLAN é um encapsulamento MAC-in-UDP, eliminando algumas das restrições descritas anteriormente. O encapsulamento permite que as sub-redes em Camada 2 se expandam para redes físicas IP em Camada 3 *stateless*. As VxLAN suportam até 16 milhões de diferentes redes virtuais (24 bits), o que de certa forma representa um número muito grande para ambientes de *Data Center* [Santana 2013]. Apesar desses melhoramentos, a VxLAN difere da nossa arquitetura na maneira como ela usa abordagem de sobreposição. A VxLAN encapsula o tráfego em um pacote UDP. Portanto, o encapsulamento impõe uma sobrecarga de 70 Bytes em cada pacote [Jeuk et al. 2014], o que não acontece com nossa arquitetura. Outra vantagem da RDNA é que realizamos a comutação dos pacotes mantendo o enquadramento Ethernet, além disso, nossos *switches* de núcleo não precisam de tabelas, ou convergência via protocolos de roteamento (BGP, EIGRP, IS-IS e OSFP) o que simplifica o desenvolvimento do *hardware* rumo a uma rede de núcleo *fabric* [Casado et al. 2012].

Abordagens SDN versus RDNA

Em [Ramos 2013] é proposto uma arquitetura denominada SlickFlow, que utiliza roteamento na origem, onde as fontes especificam um caminho no cabeçalho do pacote, contrastando com o modelo tradicional de roteamento distribuído. A argumentação do SlickFlow é que o roteamento na origem combinado com o modelo de controle centralizado do SDN/OpenFlow permite que a proposta seja empregada facilmente em diferentes topologias. Para isso, a arquitetura emprega caminhos pré-computados que levam à redução da sobrecarga na instalação dos caminhos nos comutadores intermediários, além de permitir a recuperação de falhas.

Basicamente, o SlickFlow propõe o uso de um cabeçalho contendo um percurso (segmentos), definido pelo autor como caminhos codificados ou *Encoded Paths* (EP). O EP especifica um conjunto de nós no DC através do qual os *switches* intermediários devem encaminhar os pacotes. É de responsabilidade do *switch* ToR (*Top of Rack*) a codificação dos segmentos e o mapeamento no cabeçalho do pacote. Cabe ao controlador de rede (Network Controller - NC) a tarefa de enviar as regras de fluxos (inserção e remoção dos segmentos codificados nos pacotes) [Ramos 2013].

Nossa arquitetura possui funcionalidade similar à proposta do SlickFlow. Podemos considerar equivalentes o NC com o Elemento de Gerenciamento, o primeiro *switch* no ToR com o Elemento de Borda, e por último os *switches* intermediários com o Elemento de núcleo. Todavia, a similaridade para por ai. O NC define o caminho e entrega as regras

de estado ao *switch* ToR, que precisa instalar as regras de estado nos *switches* intermediários ao longo do caminho. Nossa arquitetura não requer a instalação de nenhum estado nos comutadores de núcleo. Outra diferença está no mecanismo de encaminhamento utilizado nos *switches* intermediários. O SlickFlow exige a reescrita do pacote a cada salto para deslocamento dos *bits* no cabeçalho do pacote (*shift*). Em nossa arquitetura, pelo contrário, nenhuma reescrita é executada nos comutadores de núcleo[1]. Portanto, nosso mecanismo de encaminhamento no núcleo é mais simples. Além disso, nossa arquitetura não requer a instalação de estado para os fluxos nos *switches* de núcleo, o que simplifica o gerenciamento e a escalabilidade da rede.

Como mecanismo de recuperação à falhas, o SlickFlow sugere duas abordagens: i) usar um controlador centralizado para alterar as regras que definem as rotas na origem; e ii) utilizar caminhos alternativos nos cabeçalhos dos pacotes, tornando a arquitetura capaz de se recuperar de falhas sem a intervenção do controlador. Este caminho alternativo pode incluir proteção completa ao longo de toda rota principal. O uso do Elemento de Gerenciamento também é considerado em nossa arquitetura como modelo reativo de recuperação à falha. Todavia, apenas para tratar falhas nos caminhos emergenciais, ou seja, apenas sob condições raras e extremas utilizamos a recuperação reativa. Portanto, nossa arquitetura foi projetada para oferecer mecanismo rápido de recuperação à falha nos elementos de núcleo. Para isso, diferente dos caminhos alternativos proposto no SlickFlow, nossa arquitetura utiliza apenas um identificador de rota emergencial. Este identificador permite ao elemento de núcleo a recuperação imediata sem a dependência do controlador e sem nenhuma mensagem de controle ao longo de toda rota principal.

Na primeira abordagem o principal problema está na reação reativa, apesar de mais flexível, esta abordagem insere considerável latência, além de sobrecarregar o controlador nos momentos de falha da rede. Na segunda abordagem o problema está na complexidade para estabelecer proteção completa do caminho principal. O cabeçalho com os caminhos alternativos é representado com uma sequência de segmentos que serão utilizados por cada salto ao longo do caminho para realizar o encaminhamento. Porém, além do próximo salto do pacote, o segmento indica também mais duas informações: i) o comprimento do caminho alternativo de k e; ii) o caminho alternativo de k, onde k é representado por uma sequência de rótulos. O cabeçalho contém ainda uma informação adicional: um campo de 1 bit, que especifica se o pacote está ou não em um caminho alternativo. Esta informação adicional serve para indicar ao *switch* intermediário como tratar o pacote na ocorrência de uma falha, ou seja, se o bit possuir valor 0 indica que ele deve usar o caminho alternativo. Porém se possuir valor 1, indica que o pacote já esta em um caminho alterativo, logo, se outro falha ocorrer o pacote é descartado.

Ambas as abordagens desta arquitetura requerem a reescrita do cabeçalho dos pacotes a cada salto, além de requerem que o controlador atualize as tabelas de encaminhamento

[1]Em condições normais, ou seja, sem falha no enlace da porta de saída do pacote.

dos *switches* intermediários que pertencem ao caminho alternativo, o que dificulta o gerenciamento das DCNs.

Em nossa arquitetura o identificador de rota emergencial, assim como o identificador de rota principal, possuem comprimento máximo fixo, limitado aos 6 *Bytes* do endereço MAC de origem e destino, respectivamente. Portanto, não é necessário deslocamento variável para leitura dos identificadores de rota, o que simplifica a implementação em *hardware*. Outra vantagem é que na ocorrência de falhas na rota emergencial o pacote é redirecionado para o elemento de gerenciamento que recalcula a rota eliminando a conexão em falha. Contudo, neste cenário eventual latência adicional deve ser considerada. Isto ocorre pela necessidade de comunicação entre os planos de controle e dados, além da alteração no estado do fluxo nos elementos de borda (ingresso e egresso) para evitar a rota em falha.

Em [Martinello et al. 2014] foi proposto uma técnica de encaminhamento para Redes de Núcleo de Operadoras chamado KeyFlow. Assim como a RDNA o KeyFlow possui um controlador centralizado, o nó de ingresso modifica o cabeçalho do pacote (endereço MAC) incluindo um identificador denominado "route path ID". Cabe ao nó egresso remover o identificador, substituindo pelas informações originais, mantendo transparente esta alteração para o destinatário. Embora o KeyFlow use o TCR para calcular rotas, seu foco era em arquiteturas para rede de núcleo. A RDNA, no entanto, direcionou sua atenção para os DCs, no qual as topologias são indubitavelmente diferentes. Por exemplo, as topologias de DCNs comuns possuem dois níveis (*Spine* e *Leaf*), onde existem vários caminhos disponíveis com o mesmo tamanho. Outra diferença é que no KeyFlow não há considerações sobre implementações de *hardware*, problemas de recuperação de falhas e comunicações multicast. Além disso, nossa nova arquitetura foi implementada e validada em netFPGA SUME 10Gbps.

O *Segment Routing* [Filsfils et al. 2014], atribuído ao grupo de trabalho em redes do IETF, propõe que os pacotes sejam encaminhados usando segmentos. Esses segmentos tipicamente são anexados aos pacotes e compõe uma lista ordenada codificada, denominada pilha de etiquetas ou *stack of labels*. Para o encaminhamento dos pacotes, o *Segment Routing* precisa reescrever o ID do segmento usando operações remoção ou *pop* a cada nó do caminho (o topo da pilha é considerado o ID do segmento ativo), enquanto o MPLS executa *troca de rótulo* onde o rótulo é trocado por um novo.

Ao contrário do *Segment Routing*, o encaminhamento na arquitetura RDNA é baseado em uma operação simples de módulo, sem operações de troca ou *pop* por nó no núcleo. A RDNA permite direcionar um fluxo por meio de qualquer caminho, topologia ou encadeamento de serviço, mantendo o estado por fluxo apenas nos elementos de borda.

Outra proposta que utiliza o arcabouço das SDNs é o Path Switching [Hari et al. 2015]. Neste trabalho os autores propõem uma alternativa ao MPLS. A ideia é usar as vantagens do roteamento de origem, codificando as informações de encaminhamento em um espaço

fixo existente no cabeçalho dos pacotes. Entretanto, os autores apenas apresentam um prova de conceito em alto nível, nenhuma validação experimental ou implementação em *testbed* é explorada. Portanto, a ausência de evidências e resultados inviabiliza não apenas a comparação com nossa arquitetura, mas também a aplicabilidade do Path Switching nas Redes de *Data Centers*. Na Tabela 6.1 resumimos as principais características das tecnologias e arquiteturas discutidas neste tópico.

Tecnologias e arquiteturas	**Requer estado armazenado em tabela**	**Reescrita de cabeçalho**	**Controlador centralizado**
MPLS	Sim	Sim	Não
VLANs	Sim	Não	Não
SlickFlow	Sim	Sim	Sim
KeyFlow	Não	Não	Sim
Segment Routing	Sim	Sim	Não
RDNA	Não	Não	Sim

Tabela 6.1: Tecnologias e arquiteturas para encaminhamento de pacotes baseado em rótulos.

6.2 Roteamento Resiliente

Como primeiro mecanismo propício a suportar resiliência em DCN consideramos os protocolos de Camada MAC. A adoção do padrão Ethernet pela indústria causou a criação de grandes domínios em Camada 2 que exigem convergência rápida.

Mecanismos e protocolos de proteção a rotas em Camada 2

No melhor conhecimento deste autor, o primeiro protocolo que proporcionou redundância em redes comutadas por pacotes foi o *Spanning Tree Protocol* (STP) [Perlman 1985], este protocolo surgiu oferecendo redundância entre *bridges* e LANs. Basicamente, permitiu estender as redes locais em instâncias maiores, além de preservar a conectividade em períodos de indisponibilidade. Sua habilidade de auto-configuração representou um importante marco, mantendo a característica dos protocolos de Camada 2. O STP também inovou ao assegurar que as redes redes Ethernet, não permaneça em *loop* transiente independentes de seu tamanho.

Por outro lado, além do *broadcast*, o uso do STP introduz nas redes ineficiências substanciais como o bloqueio de alguns caminhos a fim de assegurar que não aconteçam *loops*. Outra limitação está no tempo de recuperação pós-falha. Tal ação transcorre na casa de algumas dezenas de segundos. Melhorias foram inseridas no *Rapid Spanning Tree Protocol* (RSTP), reduzindo para alguns segundos o tempo de convergência [Marchese and Mongelli 2012]. Contudo, no contexto de um problema de árvore

geradora, os caminhos redundantes são desativados, logo, o único caminho que permanece ativo se torna um gargalo na rede.

Para remover as limitações impostas por uma única árvore, o RSTP foi estendido para o *Multiple Spanning Tree Protocol* (MSTP) [Marchetti et al. 2005]. O MSTP permite que múltiplas árvores de expansão simultâneas coexistam dentro da mesma rede, mapeando cada árvore de expansão para uma instância de árvore de expansão múltipla (MSTI). Uma ou várias VLANs podem ser associadas a uma MSTI, o que permite ampliar a disponibilidade agregada da rede, além de melhorar significativamente a confiabilidade na ocorrência de falhas. Os gargalos podem ser reduzidos, distribuindo a carga da rede sobre um maior número de caminhos. Entretanto, dentro de um MSTI os integrantes continuam sujeitos a interrupções da árvore *spanning*, além do tempo de reparação. Outro problema relacionado a agregação das VLANs é que enquanto algumas VLANs podem sofrer pouca interrupção, outras VLANs podem ser severamente impactadas pois compartilham um único caminho. Além disso, como VLANs são usadas para garantir isolamento entre os grupos, o número máximo de grupos é limitado a 4096 (o número de VLANs permitido pelo padrão 802.1q).

Para automatizar o processo de configuração, melhorias foram desenvolvidas PVST+ [Pang 2007]. Todavia, o tempo de convergência permaneceu inalterado. Outro ponto que desfavorece o uso deste protocolo está no fato de somente ser suportado por equipamentos do fabricante Cisco ®. Este mesmo fabricante também desenvolveu o *EtherChannel* [Gallagher et al. 2012] que permite a agregação de um conjunto de portas físicas padrão Ethernet, criando uma única porta lógica. O objetivo da proposta é fornecer altas velocidades (Gigabit ou 10-Gigabit) e ao mesmo tempo tolerar as falhas nos enlaces ou nas portas. Uma limitação do *EtherChannel* é que todas as portas físicas do grupo de agregação devem pertencer ao mesmo comutador, exceto no caso de uma pilha de comutadores, onde podem residir em comutadores diferentes na pilha. Outra forte limitação e a característica proprietária do protocolo.

Similar ao *EtherChannel*, o LACP 802.3ad [Seifert 2000] também suporta configuração automática. A principal diferença entre os dois está no fato de um ser padrão proprietário e o outro um padrão aberto. Contudo, em se tratando de Redes de *Datacenter* dois principais fatores limitam o uso do *EtherChannel* ou LACP 802.3ad: i) as estruturas topológicas se tornariam inviáveis dado o aglomerado de cabos e conexões e ii) o desperdício de recursos sobressalentes para cada conjunto de agregação.

Outra solução proprietária da Cisco ® é o Flex Links [Donahue 2011], basicamente, um par de portas é configurado, onde uma porta torna-se o *backup* da outra. A vantagem do uso desta abordagem está na redução do tempo de reparação, quando comparada com as abordagens anteriores. Contudo, o tempo de reparação permanece alto para Redes de *Data Center*, além do fato de somente funcionar nos equipamentos do fabricante supracitado.

Ainda na Camada 2 encontramos o *Intermediate System to Intermediate System* (IS-IS) [Oran 1990, Gredler and Goralski 2005]. Este protocolo é similar ao OSPF [Moy 1998], também utiliza informação baseado nos estados dos enlaces, é um protocolo de padrão aberto e utiliza o algoritmo Dijkstra [Dijkstra 1959] para a seleção do melhor caminho na rede. IS-IS apresenta um tempo de convergência médio de 30s [Lee et al. 2004]. Apesar de ser aplicada entre sistemas de intra domínio, tradicionalmente sua aplicação não esta direcionada a conectividade no contexto desta proposta.

Resumidamente, podemos concluir que independente do protocolo de Camada 2 utilizado, um desafio comum em realizar resiliência eficiente, ou seja, no menor tempo possível, está na troca de mensagens de controle dos protocolos para prevenção de *loops* e inconsistências. Outra desvantagem deste processo é o consumo de largura de banda para o compartilhamento de informações topológicas da rede.

Mecanismos e protocolos de proteção a rotas em Camada 3

Por sua vez, protocolos de roteamento para redes de intra domínio (Camada 3), como OSPF [Moy 1998] podem ser empregados para encontrar os caminhos de menor custo entre os *hosts*. Falhas na rede em topologias de grande escala são comuns. O OSPF pode detectar essas falhas e, em seguida, transmitir as informações para todos os *switches* para assim, evitarem enlaces indisponíveis [Ramos 2013] dinamicamente. Entretanto, essa arquitetura também impõe suas limitações. A primeira limitação ao utilizar o protocolo OSPF está no tempo de convergência da rede [2], que pode demorar dezenas de segundos [Siddiqi and Nandy 2005]. Este tempo pode ser menor, reduzindo o tempo de troca de mensagens e de reação, contudo, o maior problema da arquitetura híbrida é a sua enorme sobrecarga de configuração. A atribuição de prefixos para as sub-redes, e a configuração das sub-redes nos roteadores normalmente são processos manuais, pois a atribuição deve seguir a hierarquia de endereçamento, e deve ser planejada de forma a reduzir o espaço de endereços desperdiçados, considerando também o uso futuro de endereços para minimizar uma reconfiguração posterior.

Outra proposta que funciona com as mesmas características do OSPF foi apresentada em [Wang and Crowcroft 1992]. Denominada *SPF-Emergency Exit* (SPF-EE), que é uma extensão do algoritmo SPF padrão. Nessa abordagem quando há ocorrência de congestionamento ou falhas de equipamentos, a rotina de roteamento do nó ao invés de iniciar a atualização das tabelas de roteamento, encaminha temporariamente o tráfego por caminhos alternativos. Estes caminhos alternativos são escolhidos dinamicamente a partir da troca de informações de controle entre nós vizinhos. O principal requisito da escolha do caminho alternativo, além de localizar o próximo nó vizinho que tenha um

[2] O tempo total de convergência do OSPF é dependente do tamanho da topologia, da configuração do protocolo, da quantidade de nós na rede, da velocidade de processamento do nó, da carga de tráfego, e da quantidade de prefixos de rede existentes [Barreto 2008].

caminho alternativo, é evitar *loops* de roteamento [Barreto 2008]. Na avaliação efetuada pelos autores os resultados foram considerados satisfatórios em termos de funcionalidade da abordagem (desvio do tráfego a partir de falhas transitórias/congestionamento). Entretanto, a abordagem gera um significativo tráfego adicional para encontrar e estabelecer um caminho alternativo. Além de ser lento em relação à reação global do OSPF.

6.2.1 Mecanismos de recuperação reativos e proativos

Outro protocolo que dá suporte aos mecanismos de recuperação de falhas é o OpenFlow. Ao utilizar um *switch* OpenFlow e um controlador SDN centralizado para conduzir as decisões de encaminhamento com uma visão global da arquitetura. Este modo, conhecido como reativo, permite utilizar mecanismos de recuperação dinâmicos, monitorando os enlaces da infraestrutura. É considerado mais flexível, desde que o controlador esteja habilitado para administrar a lógica que não pode ser implementada localmente no *switch*. Por outro lado, este mecanismo acrescenta latência adicional extra sobre o primeiro pacote do fluxo [OpenDaylight 2013].

O monitoramento no OpenFlow pode ser realizado usando o protocolo LLDP. Contudo, o monitoramento via LLDP apresenta sérios problemas de escalabilidade, basta ver que para aplicações que requerem recuperação menor ou igual a 50 ms será necessário monitorar os estados dos enlaces a cada 10 ms, isso significa que o controlador deve estar habilitado para receber e processar 100 mensagens a cada intervalo de 1s para cada enlace de cada comutador que compõe a rota fim-a-fim (túnel) [Kempf et al. 2012]. Dependendo da topologia e da quantidade de servidores no *Data Center*, não é incomum encontrarmos milhares de enlaces e comutadores na infraestrutura, logo, o controlador seria sobrecarregado com milhões de mensagens de controle para monitorar o estado (saúde) da rede. Isso não é apenas um desafio para os controladores, mas também para a rede como um todo, afinal, milhões de mensagens de controle irão trafegar e disputarão espaço nos enlaces com os pacotes de dados.

Outro modo de recuperação utilizando o protocolo OpenFlow é a forma proativa. Este modo permite implementar proteção com rotas pré-calculadas, através do recurso denominado *Fast Failover* (FF). Este recurso executa o primeiro *bucket* "vivo". Logo, este tipo de grupo permite ao *switch*, por exemplo, mudar a porta de encaminhamento de modo proativo, desde que já tenha sido definida está condição via controlador. Se não houver *bucket* "vivo", os pacotes são descartados. Lembrando que o *switch* que detecta a falha, usando o FF, permite a recuperação quase que instantaneamente, sem a intervenção do controlador. Contudo, não permite a notificação dos *switches* que fazem parte da rota previamente definida. Para realizar tal ação seria necessário a intervenção do controlador. Portanto, o OF (FF) não possui nenhum mecanismo de recuperação fim a fim, apenas implementa recuperação local da falha.

Enquanto que o monitoramento usando OF se apresenta como uma proposta não escalável para DCNs, executar um módulo de engenharia de tráfego em um controlador centralizado pode ser útil para o processo de balanceamento de carga, com claras consequências em função da sobrecarga e da latência inserida pelo controlador. A motivação proposta no trabalho Detour [Capone et al. 2015] é justamente utilizar menos o controlador SDN. Para isso, propõe uma abordagem denominada OpenState, cuja tarefa de controle é somente ter o conhecimento do *switch* local, não sendo necessário manusear com o controlador.

A abordagem proposta em Detour é que o controlador não precisa ser notificado no momento da ocorrência de uma falha na rede, por exemplo falha de um enlace. Segundo a proposta, somente em alguns casos é necessário ter o conhecimento global centralizado da topologia no processo de recuperação. Assim, os autores defendem como vantagem a ação de notificar somente os nós vizinhos sobre a falha, sendo que somente o primeiro pacote vai sofrer a retransmissão (retornar para o nó anterior), não sendo assim necessário propagar tal informação para o controlador. Entretanto, é possível observar uma grande deficiência no Detour, principalmente na ocorrências de falhas temporais e transitórias, comuns em infraestruturas como de um DC. Depois de configurada a rota de recuperação a proposta não monitora a rota substituída, ou seja, se os enlaces retornarem seu estado normal de funcionamento os pacotes continuarão sendo transmitidos pela rota pós falha. Logo, se esta rota não for tão eficiente como a anterior, os fluxos serão naturalmente afetados.

Em [de Lima et al. 2015], colaboramos para avaliar a flexibilidade utilizando o OpenFlow como mecanismo dinâmico de recuperação de falhas em um DCN em topologia de hipercubo. Na ocorrência de uma falha nos enlaces, um controlador SDN é notificado e modifica a forma de encaminhamento dos nós afetados por essa falha, calcula rotas alternativas e instala novas regras de encaminhamento, garantindo a entrega dos pacotes. Os resultados mostraram que o tempo não aumenta de forma diretamente proporcional em relação a quantidade de falhas. O aumento no tempo de restauração de falhas na queda de vários enlaces se dá em decorrência das ações que o controlador implementa ao detectar uma falha, mesmo que elas ocorram simultaneamente, pois para cada detecção de falha um novo grafo topológico precisa ser gerado, novas rotas são calculadas e em seguida são enviadas as mensagens para as tabelas de encaminhamento de cada nó afetado pela falha. Todas essas ações ocorrem para cada falha detectada. Todavia, vale destacar que a aplicação do plano de controle não foi implementada usando-se conceitos de multitarefa e concorrência, o que implica que as falhas simultâneas foram tratadas individualmente pelo controlador e essa característica tem um impacto direto nos resultados obtidos e serão alvo de melhoramentos em trabalhos futuros proposto pelos autores.

Na Tabela 6.2 apresentamos uma análise sintética das principais abordagens e protocolos que suportam resiliência em Camada 2 e Camada 3 focando os mecanismos de

recuperação, e o tempo de convergência.

Mecanismos de Recuperação									
Protocolo	**Pré-Calculado**	**Dinâmico**	**Proteção**	**Restauração**	**Local**	**Global**	**Centralizado**	**Distribuído**	**Tempo de Convergência**
STP (802.1D)		X		X	X	X		X	>1 minuto
RSTP (802.1w)		X		X	X	X		X	>250 ms
MSTP (802.1s)		X		X	X	X		X	>250 ms
RPVST+		X		X	X			X	>250 ms
EtherChannel	X	X	X		X			X	<250 ms
LACP 802.3ad	X	X	X		X			X	<250 ms
Flex Links	X		X		X				<150 ms
OSPF		X		X	X	X		X	40 s [Siddiqi and Nandy 2005]
IS-IS		X		X	X	X		X	30 s [Barreto 2008]
OpenFlow Reativo		X		X	X		X		100 ms [Adrichem et al. 2014]
OpenFlow Proativo	X		X		X		X		4.1 ms [Adrichem et al. 2014]
ECMP		X		X	X		X		300 ms [Iselt et al. 2004]
RDNA	X		X			X	X		600 ns

Tabela 6.2: Tempo médio do processo de recuperação de falha dos protocolos e mecanismos de recuperação. Adaptado de [Cisco 2015].

6.2.2 Reação à falha local com roteamento na origem

A técnica de roteamento na origem é a base de muitas propostas para melhorar a confiabilidade e o desempenho das redes, essencialmente porque fornece diversidade de caminhos e reduz a dependência de um único caminho de rede com características indesejáveis.

Apesar das vantagens, um problema clássico nesta abordagem é a percepção rápida para reagir a falha de um enlace ou nó que pertence a um caminho. Existem basicamente duas abordagens de alto nível para resolver o problema de reação de falha rápida. A primeira abordagem consiste em enviar uma notificação de falha para o nó de origem. Enquanto melhora o tempo de reação de falha, a origem ainda precisa aguardar para receber a mensagem de notificação. Até que a notificação de falha seja recebida, os pacotes que já haviam deixado o nó de origem serão descartados.

Na segunda abordagem a reação de falha acontece dentro da rede, usando caminhos alternativos, que é uma rota de proteção com um ou mais estados de *backup* para cada destino. Assim, um comutador pode alternar localmente para o caminho alternativo assim que o comutador detectar uma falha em um de seus enlaces diretamente conectados que afetam esse caminho. No entanto, esta abordagem requer que cada *switch* seja capaz de comutar e armazenar os caminhos de *backup*, de modo que haja uma dependência entre cada tabela de encaminhamento do *switch* e a topologia de toda a rede [Nguyen et al. 2011].

O uso de tabela de roteamento permite não apenas armazenar os estados para o encaminhamento dos pacotes sob condições normais de conectividade, mas também armazenar os estados de proteção, utilizados pelo mecanismo de recuperação à falha. Existem diversas propostas que utilizam caminhos alternativos. Dentre elas destacamos as que se relacionam com a nossa arquitetura e que poderiam ser aplicadas em DCNs, como por exemplo o MPLS Fast Reroute [Swallow et al. 2005], o Routing Deflections [Yang and Wetherall 2006], o SlickFlow [Ramos et al. 2013] e também o Path Switching [Hari et al. 2015]. A Tabela 6.3 resume as principais características dessas propostas.

A parte comum das propostas que visam a recuperação dentro da própria rede é a precomputação de caminhos alternativos para cada destino, de modo que um comutador possa alternar localmente para o caminho alternativo sem esperar pelo processo de convergência plano de controle. O SlickFlow [Ramos et al. 2013], por exemplo, propõe como solução para recuperação de falha nos comutadores intermediários com uso de caminhos alternativos codificados no cabeçalho dos pacotes. Esta abordagem exige o deslocamento dos *bits* no cabeçalho a cada salto além do uso de proteção por segmentos de caminho, o que eleva consideravelmente a codificação da rota no cabeçalho dos pacotes.

Alguns trabalhos como: KeyFlow [Martinello et al. 2014] e ALEX [Jia et al. 2014], propõem mecanismos que permitem o encaminhamento dos pacotes pelos *switches* de núcleo da rede sem utilizar tabelas de roteamento, contudo, não adotam ou descrevem mecanismos de recuperação à falha.

Propostas e arquiteturas	Proteção de Falha em um único Enlace	Proteção de Falha em múltiplos Enlaces
MPLS Fast Reroute [Swallow et al. 2005]	Sim	Sim
Routing Deflections [Yang and Wetherall 2006]	Sim	Sim
Slick Packets [Nguyen et al. 2011]	Sim	Sim
SlickFlow [Ramos et al. 2013]	Sim	Sim
Path Switching [Hari et al. 2015]	Sim	Não
RDNA	Sim	Sim

Tabela 6.3: Síntese das principais diferenças entre as propostas que utilizam roteamento de origem sob a ótica da proteção de falha no enlace.

Em [Gomes 2016] colaboramos para uma proposta denominada KAR (Key-for-Any-Route). O KAR propõe um novo sistema de roteamento para redes de intra domínio com suporte a mecanismos de rápida reação a falhas, sem nenhuma necessidade de sinalização ou mesmo modificação do pacote ao longo do caminho. Tal proposta combina os benefícios do roteamento controlado na origem com mecanismos adicionais de deflexão guiada para fornecer resiliência através de uma diversidade de caminhos, conhecido como (*for-any-route*). Uma rota dentro do sistema de roteamento KAR é representada por um número inteiro, chamado de identificador de rota ou (*route ID*), que é codificado no cabeçalho do pacote no endereço MAC de destino. Este valor inteiro representa o caminho que o pacote vai seguir entre um par de nós (origem e destino). Resumidamente, o sistema de encaminhamento é baseado no resultado da divisão entre o identificador de rota (localizado no cabeçalho do pacote) com um identificador que é configurado em cada *switch* da topologia, exceto os *switches* de borda que são responsáveis por inserir e retirar os identificadores de rota codificados. A técnica de codificação de rota no KAR é projetada explorando uma propriedade especial do Sistema de Resíduo Numérico, ou *Residue Number System* (RNS) [Garner 1959, Chokshi et al. 2009, Chang et al. 2015].

Como mecanismo de reação rápida a falha os comutadores de núcleo do KAR fazem a deflexão dos pacotes ao invés de descartá-los. Como prova de conceito três técnicas de deflexão foram propostas e avaliadas, sendo uma totalmente aleatória utilizada como pior caso. Os pacotes que são defletidos, então, passam através de um conjunto de diversos *switches* que guiam os pacotes usando o mesmo identificador previamente codificado no cabeçalho do pacote. Esse identificador permite o transporte guiado do pacote livre de *loops* não intencionais e sem qualquer reescrita de pacotes no núcleo da rede. Assim, a abordagem do KAR mantém a conectividade dos enlaces permitindo, em tempo real, encaminhar os pacotes ao longo do caminho até o destinatário mesmo com falha em um ou mais enlaces da topologia.

Para reconduzir os pacotes de volta à rota no KAR, os nós vizinhos do caminho

principal [3] são incluídos no cálculo do que chamamos de identificador de rota (*route ID*). Assim, se o roteador de núcleo encontrar a porta do caminho principal indisponível, o pacote será defletido, ou seja, enviado para qualquer outra porta disponível exceto a porta de entrada com a mesma probabilidade. Como o *route ID* já foi codificado incluindo os respectivos vizinhos cada pacote possui a informação que permite a recuperação da falha.

Diferente da nossa arquitetura, o KAR utiliza apenas um identificador de rota (*route ID*), que se mantém inalterado mesmo em períodos de falha. Todavia, a aleatoriedade usada na deflexão como mecanismo de recuperação à falha vai resultar em um grande desafio se aplicado as DCNs com topologia 2-tier Clos Network. Por exemplo, com 24 *switches* em uma 2-tier Clos Network ($Spine = 8$ e $Leaf = 16$) com 96 portas na RDNA precisamos no pior caso de 10 Bytes (IRP + IRE). Para o KAR neste mesmo cenário é necessário no máximo a mesma quantidade de Bytes (Equação 4.1). Porém, a principal dificuldade é lidar com a aleatoriedade na camada *Leaf*, ou seja, se os pacotes forem defletidos para portas que estão conectados os *hosts*, eles podem ser reencaminhados de volta para o mesmo *switch Leaf*. Entretanto, como o *router ID* não foi alterado, o pacote será direcionado para a mesma porta, e se esta porta continuar inválida a deflexão será utilizada e novamente o mesmo problema poderá ocorrer com uma probabilidade de 92,6%[4]. Uma forma de reduzir essa possibilidade é encaminhar o pacote para o controlador que calculará um novo *route ID* excluindo o enlace defeituoso, porém essa ação certamente representará incremento de latência na entrega do pacote, além de sobrecarregar o controlador em momentos de falha da rede.

No caso de falha de enlace, uma abordagem tradicional é restauração de rota. Esta abordagem consiste em notificar o controlador, que recalcula a rota excluindo o enlace defeituoso dos caminhos disponíveis. O problema é como reagir rapidamente após uma detecção de falha, evitando a latência intrínseca para se comunicar com o controlador. Um mecanismo típico para reação rápida a falhas é proteção de rota através da deflexão dos pacotes. As técnicas de roteamento de deflexão são conceitualmente simples e permitem que cada *switch* decida independentemente quais pacotes encaminhar para qualquer enlace disponível [Yang and Wetherall 2006]. Portanto, quando um *switch* detecta uma falha em um de seus enlaces (ou seja, a porta de saída está com falha), ele escolhe uma de suas portas saudáveis para encaminhar os pacotes. A escolha das portas disponíveis é aleatória (por exemplo, distribuição uniforme) e leva os pacotes a rotas inesperadas, isto é, caminhos mais longos, como os resultados apresentados pela proposta KAR. Diferente da RDNA, que utiliza um robusto mecanismo de reação à falha, que permite encaminhar os pacotes por rotas determinísticas, além disso, nenhuma mensagem de controle é requerida.

[3] Caminho principal na abordagem KAR é definido como o menor caminho entre a origem e o destinatário, definido pelo algoritmo de Dijkstra [Dijkstra 1959].

[4] $(96 - (1 + 7))$ 96 representa a quantidade total de portas na camada *Leaf*; 1 representa a porta de entrada (NIP) que não será considerada e 7 representa o total de portas dos *switches Spine* excluindo a porta inválida que já foram consideradas no cálculo do *router ID*. Portanto teremos $P = \frac{96-8}{96-1} = 0.926$.

Em [Yang and Wetherall 2006] foi apresentada outra proposta que utiliza o roteamento por deflexão, denominada de *Routing Deflections*. Por ser uma técnica de roteamento probabilística, *loops* transitórios podem ser formados. A RDNA utiliza roteamento resiliente determinístico baseado em um mecanismo de proteção de rede ao longo de todo o percurso. Nosso mecanismo de reação a falhas rápidas usa um identificador de rota de emergencial (IRE). O IRE é calculado de forma a representar um conjunto de *switches* necessários para contornar uma falha na rota principal. Este recurso é importante e desejável em redes de *Data Center*, principalmente pelo alto grau de conectividade comumente encontrado nas topologias.

Para finalizar, inspirado em um abordagem de proteção híbrida, onde os segmentos individuais de um caminho são protegidos [van Asten et al. 2014], a RDNA inovou ao oferecer um mecanismo de reação rápido à falha não encontrado na literatura, resultando em uma abordagem de proteção completa da rota principal utilizando apenas um único identificador emergencial.

6.3 Comunicação Multicast

Cabe aos algoritmos de roteamento e técnicas de encaminhamento de pacotes utilizadas na comunicação multicast apoiar os padrões um-para-muitos, muitos-para-um e muitos-para-muitos. Os operadores de DCNs podem usar multicast IP nativo [Moy 1994, Pansiot 2010, Li and Freedman 2013]. No entanto, essa solução não é eficaz porque os *switches* podem não reconfigurar grupos multicast IP nas taxas necessárias [Shahbaz et al. 2018].

Dispositivos de rede tradicionais que encaminham tráfego (por exemplo, roteadores e *switches* de Camada 3) executam funções de roteamento e encaminhamento ao mesmo tempo. O roteamento envolve determinar o melhor caminho para um determinado pacote com base no endereço de destino e nas informações de conectividade existentes. Por outro lado, o encaminhamento é responsável por transferir um pacote de uma interface de entrada para uma de saída, conforme determinado na etapa de roteamento. Portanto, o desempenho de um roteador ou *switch* seria melhorado se ele pudesse dedicar todos os seus recursos ao encaminhamento. Ao explorar esta ideia, o conceito de SDN permitiu o surgimento de novas abordagens incluindo propostas para comunicação multicast [Islam et al. 2018].

O surgimento de redes programáveis e mais especificamente SDN com OpenFlow, tornou possível a implementação e o melhoramento de vários recursos e serviços de rede, incluindo a comunicação multicast. Maiores detalhes sobre as propostas de comunicação multicast com SDN podem ser encontradas em [Islam et al. 2018]. Entretanto, conforme discutido anteriormente, propostas de SDN tradicionais com o OF enfrentam uma série de desafios práticos quando aplicadas às DCNs. Na comunicação multicast a manutenção dos estados nas tabelas de fluxo é um dos principais limitadores da arquitetura SDN.

[Jia 2014] propôs uma abordagem que utiliza as propriedades do Teorema Chinês do Resto para representar um mapa de bits que especifica as portas de saída dos pacotes em cada *switch* de núcleo, chamada de *Code-Oriented eXplicit multicast* (COXcast). Basicamente, este mapeamento de bits é a conversão de um número decimal para binário. O número decimal obtido do resto da divisão entre um identificador específico de canais multicast e uma chave (número primo) única que identifica cada *switch* no núcleo da rede. Por exemplo, imagine uma transmissão multicast, cuja rota passa pelo *switch* com Id = 31 que possui 5 portas físicas numeradas de 0 até 4. Suponha neste exemplo que os pacotes desta transmissão deverão ser encaminhados pelas respectivas portas físicas 2 e 4. Portanto, a representação binária desta mapeamento em bits das portas deverá ser [10100], o valor 1, mais a esquerda representa a porta 4. Assim, para se obter a representação binária é necessário que o valor decimal seja 20. Usando o Teorema Chinês do Resto o valor de residual que representa o decimal requerido é 148.665 (148.665 módulo 31 = 20). A proposta deste trabalho suporta transmissões multicast e unicast, seguindo os mesmos princípios de conversão do resíduo decimal para valor binário.

A proposta defendida em [Jia 2014] utiliza o resto da divisão como um mapeamento de bits, utilizados para as portas de saída do pacote. O desafio desta abordagem é que os identificadores dos *switches*, precisam iniciar-se com valores enormes. Isto ocorre devido ao teorema da divisão euclidiana, este indica que um restante r de uma computação de divisão é um número inteiro tal que $0 \leq r < b$, onde b é o divisor, logo, como os identificadores utilizados nos *switches* são números primos/coprimos ao longo da rota multicast, estes precisam ser maiores do que o número decimal desejado. Portanto, em uma rede com *switches* que possuem 96 portas físicas o número do identificador dos switches do núcleo precisa ser maior do que 2^{96}. Resumidamente, o comprimento do identificador de rota multicast pode ser definido por: $comprimento\_bits(R) = \lceil log_2(M) \rceil$, onde M é o resultado da multiplicação dos identificadores dos *switches* ao longo da rota multicast. Em nossa arquitetura o uso das expressões polinomiais reduzem consideravelmente o tamanho do cabeçalho multicast, conforme apresentamos na Seção 4.3.

Uma outra solução multicast que poderia ser aplicada em nossa arquitetura é utilizar o controlador RDNA para inserir regras de fluxo específicas nos elementos de borda para transmitir cópias dos pacotes com diferentes identificadores de rota, por exemplo, utilizando os recursos de grupo do OpenFlow (*Group Table feature "all"*). Portanto, a partir dos elementos de borda, cópias do pacote podem ser distribuídos pelos *switches* de núcleo no *Data Center*. Todavia, observe que ao usar o *switch* de borda gerando cópias dos pacotes não oferecemos a redução do tráfego na rede. Por exemplo, em um grupo contendo 10 membros é necessário encaminhar, via enlace do *switch* de borda, 10 cópias para a camada $Leaf$. Cada pacote pode ser codificado para rotas distintas, mas de qualquer forma serão 10 cópias a partir da raiz da árvore. Portanto, utilizar o *switch* de borda não é uma solução viável em termos de redução de tráfego na rede aplicado em transmissões

multicast.

Propostas tradicionais de roteamento multicast via IP [Moy 1994, Deering et al. 1994, Ballardie et al. 1993, Waitzman et al. 1988] e alternativas como: *Bit Indexed Explicit Replication (BIER)* [Giorgetti et al. 2017, Chen et al. 2018] e *bloom filters* [Jokela et al. 2009, Ratnasamy et al. 2006] exigem estados armazenados nos *switches* intermediários, além de reescrita dos pacotes a cada salto.

Em [Shahbaz et al. 2018], os autores propõem um mecanismo nativo que aproveita as características topológicas das DCNs, tais como simetria da topologia (Fat-tree), menor caminho e a posição das VMs dos inquilinos. A partir destas características, a arquitetura Elmo codifica parte da árvore multicast dentro do próprio pacote em um cabeçalho específico (encapsulamento). Essas informações codificadas, são apoiadas por estados armazenados nas tabelas dos *switches* intermediários, e representam a árvore de multicast dentro do *Data Center*. Segundo os autores, a codificação de parte da árvore dentro do próprio pacote reduz a necessidade de armazenamento de estados nos *switches*, suportando 1 milhão de grupos multicast com apenas 1.1 mil estados de fluxos nas tabelas.

A arquitetura Elmo, utiliza um controlador centralizado que é responsável pelos estados em todos os *switches* intermediários e *hypervisors*. Também é de responsabilidade do controlador a sincronização entre as regras inseridas nos pacotes (*p-rules*) e os estados armazenados nos *switches* (chamado pelos autores de *s-rules*). Apesar da proposta Elmo apresentar razoável escalabilidade e ter sido implementada por meio de plano de dados programável (P4 [Bosshart et al. 2014]), esta arquitetura apresenta uma série de desvantagens quando comparada com nossa proposta. Por exemplo, a RDNA não precisa de nenhum armazenamento de estado nos *switches* intermediários (*Spine* e *Leaf*) o que reduz consideravelmente a sobrecarga nos controladores. Também não precisamos de notificações ou aguardar mensagens para recuperação em caso de falha. Além disso, nossa codificação de árvore multicast é extremamente simples (*bitmap* das portas de saída), diferente da proposta pela Elmo que exige complexo algoritmo para a criação dos grupos multicast.

Nossa arquitetura de roteamento multicast foi projetada considerando as seguintes premissas: i) não ser necessário armazenar estado nos *switches* de núcleo, logo, o projeto do *hardware* pode ser simplificado, além de manter a compatibilidade com nosso mecanismo de recuperação ultra-rápida no núcleo da rede; ii) visando a redução da latência de comutação do pacote, não existe reescrita do pacote a cada salto. Portanto, com essas duas premissas buscamos uma solução mais eficiente de transmissão multicast em redes de *Data Center*, além de ser escalável, conforme apresentamos em nossa análise de escalabilidade (Capítulo 4).

A Tabela 6.4 resume as principais diferenças entre algumas abordagens multicast.

Protocolo	COXcast	Multicast Tradicional	RDNA
Estado de Roteamento no Switch	Não	$\mathcal{O}$(SG)	Não
Estado de Controle no Switch	Não	$\mathcal{O}$(IG)	Não
Estado de Roteamento na Origem	Sim	Não	Sim
Controle de Estado na Origem	$\mathcal{O}$(DGR)	$\mathcal{O}$(G)	$\mathcal{O}$(G)
Endereçamento	Unicast e Multicast	Multicast	Unicast e Multicast
Escalabilidade (# de grupos)	Enorme (∞)	Pobre ($\leq$1.5k)	Enorme (∞)
Escalabilidade (tamanho do grupo)	Média[1] ($\leq$8k)	Enorme (∞)	Enorme (∞)
Tempo de Processamento	$\mathcal{O}(M)$	$\mathcal{O}(G)$–$\mathcal{O}(SG)$[2]	$\mathcal{O}(M)$
Overhead no Pacote	Enorme ($\mathcal{O}$(M))	Baixo ($\mathcal{O}(1)$)	Enorme ($\mathcal{O}$(M))
Custo da Junção (grupos)	$\mathcal{O}(D \log M)$	$\mathcal{O}(RD)$	$\mathcal{O}(D \log M)$
Tipo do Pacote	Proativo	Reativo	Proativo
Segurança	Alta	Média[1,2]	Alta
Tempo de Convergência	Baixo	Alto	Baixo
Entrega Ótima (STP)	Sim	Não	Sim
Problema Assimétrico	Não	Sim	Não
Modificação do cabeçalho	Não	Sim	Não
Suporte a Multipath	Sim	Não	Sim

Tabela 6.4: Adaptada de [Jia 2014], **G:** # de Grupos ativos; **S:** # de origens; **D:** # de receptores; **I:** # de interfaces, **R:** # de *switches* na árvore, **E:** # de *switches* de borda, **M:** Tamanho dos identificadores, [1] Dependente da Arquitetura do DC, e [2] Dependente do protocolo.

6.4 Considerações Finais

Neste capítulo descrevemos e comparamos alguns trabalhos encontrados na literatura, que podem ser aplicadas nas DCNs. Todas elas propõem soluções variadas para os desafios citado no Capítulo 1. Outras arquiteturas como: Monsoon [Greenberg et al. 2008b], VL2 [Greenberg et al. 2009] e PortLand [Niranjan Mysore et al. 2009] e propostas como Presto [He et al. 2015] e PathSets [MacDavid et al. 2017], avançaram no estado da arte, resolvendo um conjunto de desafios, como por exemplo, estimulando a comoditização com-

pleta da infraestrutura como forma de obter escalabilidade e baixo custo, simplificaram o balanceamento de carga, maximizando o uso dos enlaces disponíveis, além de melhoras significativas dos mecanismos de roteamento, encaminhamento e resiliência no DC. Todavia, todas estas arquiteturas não abordam diretamente a proteção completa da rota principal e o suporte a roteamento multicast, portanto, por estes motivos não incluímos as arquiteturas supracitadas em nossas discussões.

Observe que consideramos principalmente a descrição das técnicas de encaminhamento utilizadas, o funcionamento dos mecanismos de recuperação à falha e como cada arquitetura pode suportar o crescimento dos estados nos *switches* intermediários. A parte comum de todas as arquiteturas relacionadas está no uso do roteamento de origem. O uso desta técnica permite retirar dos *switches* e roteadores a decisão de roteamento, aumentando a escalabilidade do plano de dados, princípio este fundamental para as redes de *Data Center*.

Capítulo 7

Conclusão e Trabalhos Futuros

Vivemos cada dia mais conectados, inovações tecnológicas não assustam como antes. A Computação em Nuvem não é um conceito novo, mas nos últimos anos tem impulsionado diversas áreas, oferecendo soluções em um modelo de serviços, tais como: *Software* como Serviço (SaaS - *Software as a Service*), Plataforma como Serviço (PaaS - *Platform as a Service*) e Infraestrutura como Serviço (IaaS - *Infrastructure as a Service*) [Mell and Grance 2011]. Com base em suas necessidades, os clientes podem escolher um desses serviços para criar entidades lógicas, tais como: servidores web, unidades de armazenamento, serviços bancários, ou um computador propriamente dito. Estas entidades podem crescer ou encolher em tempo real a fim de garantir os níveis desejados de latência, desempenho, escalabilidade, confiabilidade e segurança, para atender as exigências de uma aplicação específica. Dessa forma, a computação em nuvem pode reduzir significativamente o capital inicial e tornar as despesas operacionais proporcionais às demandas atuais, tanto no quesito armazenamento, quanto no quesito processamento [Vassoler 2015].

Recentemente, novos modelos de serviço tem exigido infraestrutura mais eficientes dos provedores. Serviços emergentes de redes móveis de próxima geração 5G e Internet das Coisas são exemplos desses serviços, que requerem baixa latência e alta confiabilidade. Para atender essa nova demanda, pequenos centros de processamento e armazenamento têm sido utilizados, essa infraestrutura promete reduzir a latência e aumentar a vazão, aproximando os recursos de nuvem dos usuários finais. São exemplos os serviços de IoT, nas mais diversas áreas desde planejamento urbano, produção agrícola e industrial, logística, comércio eletrônico e até preservação do meio ambiente. De modo geral, a IoT busca facilitar nas tarefas do dia a dia, além claro de melhorar a vida das pessoas e comunidades. Seguramente várias outras novas aplicações, nas mais diversas áreas do conhecimento irão surgir, todas em um curto espaço de tempo, contudo, especialmente na área de saúde e segurança um vasto campo de oportunidades está aberto. Grandes empresas como: Intel, Google, Microsoft e Amazon estão atentas a essas demandas, investindo pesado em serviços e produtos para melhorar a qualidade de vida das pessoas.

Durante o desenvolvimento desta tese, buscou-se solucionar os problemas relacionados

a infraestruturas de DC. O objetivo foi atender de forma satisfatória as organizações e usuários da Computação em Nuvem. Pequenos centros de Computação em Nuvem, construídos em topologia de rede 2-tier, localizados próximo aos usuário, ajudaram a reduzir a sobrecarga em termos de latência e tempo de resposta da nuvem pública tradicional. Diferente de todas as arquiteturas existentes, a RDNA fornece em conjunto baixa latência na comunicação unicast e um mecanismo de reação à falha ultra-rápido, ademais também demonstramos que a RDNA permite a comunicação multicast com baixa sobrecarga e escalabilidade sem mudanças significativas.

Os resultados alcançados nesta tese se apresentaram extremamente promissores e incluem a introdução do conceito de uma Arquitetura Definida por Resíduos para Rede de *Data Centers* (RDNA). Neste contexto, consideramos à capacidade de programar as decisões de roteamento, através de identificadores codificados no cabeçalho dos pacotes para uma rede de *switches* de núcleo sem tabela. A dissociação entre o *switch* de borda, *switch* de núcleo e o controlador da rede, permite a criação de blocos fundamentais para um padrão de design SDN pragmático.

O projeto e implementação da arquitetura, demonstrando a viabilidade do RDNA através de experimentos em ambiente emulado (Mininet) e *testbed* 10 Gpbs. No melhor conhecimento do autor desta tese, esta é a primeira implementação de uma proposta SDN sobre uma rede de núcleo em NetFPGA SUME 10G, utilizando resíduos programáveis.

Nossa arquitetura é independente de topologia, sobretudo, evidenciamos sua escalabilidade para redes de DC construídas sobre 2-tier Clos *networks*. Além disso, os resultados obtidos em nosso *testbed* mostram que nossa arquitetura alcança baixa latência com o tempo de computação do pacote de ≈ 0.6 μs por salto, e sem variabilidade na rede de núcleo. Em uma rede de domínio RDNA, uma política pode ser definida, por exemplo com proteção de conectividade para serviços ou **fluxos** específicos dentro do DC. O controlador RDNA instrui os *switches* de borda de forma programável para proteger os fluxos. Desse modo, a RDNA oferece proteção no transporte *host* a *host* com reação à falha ultra-rápida (microssegundos) e sem *overhead* de mensagens de controle no núcleo do DC.

Finalmente, pode-se concluir que os resultados obtidos na avaliação da arquitetura RDNA demonstrou que os objetivos propostos foram atingidos. O TCR aplicado no sistema de roteamento e encaminhamento para DCNs se mostrou eficiente, suportando o crescimento dos estados de encaminhamento ao manter os *switches* de núcleo da topologia sem tabela. Os estados são restritos apenas aos *switches* de borda, não existindo restrição significativa em termos de capacidade de armazenamento das regras de fluxo. O mecanismo de encaminhamento é simples, baseado em uma operação de resto de divisão (módulo), que resultou em um modelo de encaminhamento de pacote na comunicação unicast com baixa latência, Ademais, o mecanismo projetado, implementado e avaliado de reação à falha forneceu um esquema de proteção completa da rota principal, mantendo

o núcleo sem tabela e sem depender de nenhuma mensagem de controle ou protocolo de convergência. Na comunicação multicast a arquitetura RDNA conquistou notáveis benefícios. Construído a partir da união de duas subáreas da Teoria dos Números, permitiu representar a árvore de multicast de forma eficaz, reduzindo em até 50% o tamanho do cabeçalho para os casos avaliados (Capítulo 4). Do mesmo modo, nenhum estado armazenado em tabela ou reescrita do cabeçalho a cada salto é requerido, o que simplifica o modelo de roteamento e encaminhamento para este tipo de comunicação. Todas essas melhorias conduzem a uma infraestrutura promissora para redes de *Data Centers* e, portanto, favorecem as aplicações e serviço atuais além de suportar de forma escalável as futuras demandas por conectividade na Computação em Nuvem.

Outras contribuições diretamente relacionadas com a arquitetura são listadas em ordem cronológica inversa através das seguintes publicações:

1. **LIBERATO, A. B.**; MARTINELLO, M.; GOMES, R. L.; BELDACHI, A. F.; HUGUES-SALAS, E.; VILLACA, R.; RIBEIRO, M. R. N.; KANELLOS, G.; NEJABATI, R.; GORODNIK, A.; SIMEONIDOU, D. RDNA: Residue-Defined Networking Architecture Enabling Ultra-Reliable Low-Latency Datacenters. Artigo submetido em 16 de abril e aprovado em 16 de outubro de 2018, IEEE Transactions on Network and Service Management, special issue in Novel Techniques for Managing Softwarized Networks, no tópico de Network softwarization for 5G e Design of architectural building blocks for managing virtualized software-defined systems.

2. MARTINELLO, M.; **LIBERATO, A. B.**; BELDACHI, A. F.; KONDEPU, K.; GOMES, Roberta L.; VILLACA, R.; RIBEIRO, M. R. N.; YAN, Y.; HUGUES-SALAS, E.; SIMEONIDOU, D. Programmable Residues Defined Networks for Edge Data Centres. 13th International Conference on Network and Service Management (CNSM, 2017), Tokyo, Japan, Nov. 26-30, 2017.

3. GOMES, R. R.; DOMINICINI, C. K.; **LIBERATO, A. B.**; RIBEIRO, M. R. N.; MARTINELLO, M. Analytical Modeling Approach of Routing Deflection for Intradomain Networks In: XXXVI Congresso da Sociedade Brasileira de Computação, 2016, Porto Alegre/RS. XV Workshop em Desempenho de Sistemas Computacionais e de Comunicação, 2016.

4. **LIBERATO, A. B.**; MARTINELLO, M.; RIBEIRO, M. R. N.; MARQUEZ-BARJA, J. M.; KAMINSKI, N.; DASILVA, L. A. Dynamic Backhauling within Converged Networks In: ACM SIGCOMM, 2016, Florianópolis. ACM SIGCOMM Workshop on Fostering Latin-American Research in Data Communication Networks (LANCOMM), 2016.

5. GOMES, R. R.; **LIBERATO, A. B.**; DOMINICINI, C. K.; RIBEIRO, M. R. N.; MARTINELLO, M. KAR: Key-for-Any-Route, a Resilient Routing System In: The

46th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'2016), 2016, Toulouse-France. The 2nd Workshop on Dependability Issues on SDN and NFV (DISN), 2016.

6. LIMA, D. S. A.; GUIMARAES, R.; **LIBERATO, A. B.**; SPALLA, E. S.; VASSOLER, G. L.; MARTINELLO, M.; VILLACA, R. REUNI: Um algoritmo para REduzir tabelas de encaminhamento e UNIformizar a distribuição dos fluxos em redes com topologia hipercubo In: XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), 2016, Salvador-Bahia. XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), 2016.

7. MAFIOLETTI, D. R.; **LIBERATO, A. B.**; DOMINICINI, C. K.; VILLACA, R.; MARTINELLO, M.; RIBEIRO, M. R. N. Latency Measurement as a Virtualized Network Function using Metherxis. ACM SIGCOMM Computer Communication Review. https://ccronline.sigcomm.org/, 2016.

8. MAFIOLETTI, D. R.; **LIBERATO, A. B.**; DOMINICINI, C. K.; VILLACA, R.; MARTINELLO, M.; RIBEIRO, M. R. N. Metherxis: Virtualized Network Functions for Micro-second Grade Latency Measurements In: ACM SIGCOMM, 2016, Florianópolis. ACM SIGCOMM Workshop on Fostering Latin-American Research in Data Communication Networks (LANCOMM), 2016.

9. LIMA, D. S. A.; GUIMARAES, R.; VASSOLER, G. L.; MARTINELLO, M.; VILLACA, R.; **LIBERATO, A. B.** Avaliação do Uso do OpenFlow na Recuperação de Falhas em Data Centers Centrados nos Servidores In: XIV Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance 2015), 2015, Pernambuco-Recife. XXXIV Congresso da Sociedade Brasileira de Computação – CSBC, 2015.

10. **LIBERATO, A. B.**; MAFIOLETTI, D. ; SPALLA, E. ; VILLACA, R. ; MARTINELLO, M. Avaliação de Desempenho de Plataformas para Validação de Redes Definidas por Software. In: Wperfomance, 2014, Brasília. WPerformance - XIII Workshop em Desempenho de Sistemas Computacionais e de Comunicação, 2014.

11. **LIBERATO, A. B.**; SPALLA, E.; MAFIOLETTI, D.; VILLACA, R.; MARTINELLO, M. Balanceamento de Carga em SDN Provido por Comutadores Estocásticos de Prateleira. In: IV Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF), 2014, Florianópolis. Simpósio Brasileiro de Redes de Computadores - SBRC, 2014.

As seguintes publicações (listadas em ordem cronológica inversa) não estão diretamente conectadas ao tema desta proposta. Contudo foram favoráveis à construção do conhecimento e portanto devem ser consideradas também contribuições relevantes.

1. SPALLA, E. S.; MAFIOLETTI, D. R.; **LIBERATO, A. B.**; VILLACA, R.; ROTHENBERG, C. E.; CAMARGOS, L.; EWALD FILHO, G.; MARTINELLO, M. AR2C2: Actively Replicated Controllers for SDN Resilient Control Plane In: IEEE/IFIP Network Operations and Management Symposium, 2016, Istanbul. IEEE/IFIP Network Operations and Management Symposium, 2016.

2. SPALLA, E.; MAFIOLETTI, D.; **LIBERATO, A. B.**; Rothenberg C.; CAMARGOS, L.; VILLACA, R.; MARTINELLO, MAGNOS. Estrategias para Resiliência em SDN: Uma Abordagem Centrada em Multi-Controladores Ativamente Replicados. In: XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2015, Vitoria-ES. Anais do SBRC, 2015.

3. SPALLA, E.; MAFIOLETTI, D.; **LIBERATO, A. B.**; VILLACA, R.; MARTINELLO, M. Resiliência no Plano de Controle para Redes Definidas por Software. In: IV Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF), 2014, Florianópolis. Simpósio Brasileiro de Redes de Computadores - SBRC, 2014.

4. MARTINELLO, M.; VILLACA, R.; **LIBERATO, A. B.**; MAFIOLETTI, D.; SPALLA, E.; CABELINO, R. Plataformas Abertas para Infraestruturas Definidas por Software: Projeto, Implementação e Experimentos. In: Infraestruturas Definidas por Software (IDS) no WRNP, 2014, Florianópolis. Workshop da Rede Nacional de Ensino e Pesquisa (WRNP), 2014.

7.1 Trabalhos Futuros

A RDNA contribuiu para o estado da arte através de uma arquitetura original que fornece resiliência na comunicação unicast além de uma abordagem de roteamento e encaminhamento escalável para comunicação multicast. Todavia, no período de pesquisa, projeto e implementação desta tese observamos diversas outras oportunidades de pesquisa e algumas limitações que poderão ser investigadas futuramente.

Uma dessas oportunidades está em implementar um algoritmo de balanceamento de carga para distribuição dos fluxos entre todos os caminhos disponíveis na topologia, a partir de um par (origem e destino). Atualmente, a RDNA depende da decisão do controlador que calcula diferentes identificadores para rota principal e emergencial. Esses identificadores são enviados do controlador para o *switch* de borda, que é responsável pela codificação nos campos existentes antes de encaminhar o pacote. Portanto, uma forma de realizar o balanceamento de carga seria calcular um conjunto de identificadores de rota (principal e emergencial) via controlador e entregar uma lista com este conjunto de caminhos para os *switches* de borda (de forma proativa ou reativa). Este agrupamento poderia ser realizado via endereço IP entre os pares (origem e destino). Assim o próprio *switch* de borda já po-

deria distribuir os fluxos entre todos os múltiplos caminhos disponíveis, sem depender do controlador. Esta distribuição poderia considerar todos os caminhos de forma igualitária, implementando uma lista circular simples "round-robin (RR)", por exemplo. Deste modo, os fluxos poderiam ser melhor distribuídos sem mensagens de controle entre o *switch* de borda e o controlador. Esse, contudo, manteria a flexibilidade permitindo que essa lista fosse atualizada a qualquer instante. Outras abordagens mais sofisticadas poderiam considerar inclusive a classificação do fluxo em função de seu tamanho, auxiliando o plano de dados da arquitetura RDNA, por exemplo, reservando um caminho para fluxos elefantes, assim todos fluxos com essa classificação poderiam disputar este único caminho sem prejudicar os fluxos menores.

Atualmente uma das limitações da RDNA é que não consideramos o tratamento de prioridades nas filas para garantia de Qualidade de Serviço (QoS). Portanto, a implementação de mecanismos que auxiliem a QoS é uma importante melhoria para a arquitetura RDNA. Consideramos que a associação de serviços de QoS para diferentes classes de tráfego, conforme utilizado em [Oliveira et al. 2018], poderá atender a esta demanda. Deste modo, um próximo passo importante é avaliar o impacto no comprimento do cabeçalho para a comunicação unicast, e a partir desse resultado avaliar a integração com nossa arquitetura.

No Capítulo 5 demonstramos a eficiência da arquitetura RDNA versus SDN OpenFlow tradicional. Entretanto, acreditamos que é possível melhorar nossos resultados em termos de latência de encaminhamento paralelizando o cálculo nas operações de resíduo, inclusive no mesmo ciclo de (*clock*) [Nunez-Yanez et al. 2016]. Outro aspecto é a economia de energia, eliminando-se a TCAM e explorando técnicas de adaptação na frequência do *clock* para projetar *switches* com consumo de energia proporcional à demanda de tráfego. Os trabalhos desenvolvidos em [Cercós et al. 2015, Nunez-Yanez et al. 2016] podem auxiliar nesta linha de investigação.

Soluções de roteamento e encaminhamento multicast são tipicamente complexas e limitadas em termos de comprimento (tamanho) de seus cabeçalhos. A RDNA avançou no estado da arte com relevante redução no tamanho do cabeçalho sem perder a flexibilidade exigida por este tipo de comunicação, além disso, nossa arquitetura não requer reescrita do cabeçalho a cada salto sequer estado armazenado em tabela nos *switches* de núcleo. Entretanto, uma possível limitação da nossa abordagem esta no cálculo do polinômio específico utilizado. Sabe-se que a complexidade de tempo computacional da função algébrica utilizada no polinômio de grau n com coeficientes de tamanho fixo é $\mathcal{O}(n)$ [Knuth 1998]. Deste modo, dependendo da quantidade de portas do *switch*, do tamanho da topologia e do algoritmo usado para calcular a expressão polinomial, pode-se inserir nos *switches* de núcleo certa sobrecarga de processamento, resultando no incremento de latência para o encaminhamento dos pacotes na comunicação multicast. Portanto, a implementação e experimentação do roteamento e encaminhamento RDNA multicast é um tema a ser

investigado como consequência desta tese.

Em relação a implementação e experimentação do roteamento e encaminhamento RDNA multicast consideramos que os avanços do plano de dados programáveis, conforme discutido em: [Bosshart et al. 2013] baseado em SDN e *hardware* (FPGA), propostas utilizando P4 [Bosshart et al. 2014, Sivaraman et al. 2015] e Protocol Independent Switch Architecture (PISA) como Barefoot Tofino [Barefoot 2018], além do Protocol Oblivious Forwarding (POF) [Li et al. 2017], permitirão a implementação e a validação da nossa abordagem.

A partir dos resultados obtidos nesta tese, e da crescente demanda da Internet das Coisas, 5G e Indústria 4.0, acreditamos que a RDNA possui características únicas que permitirão atender a demanda de serviços de conectividade para *Data Centers* de borda, preferencialmente localizados próximos aos usuários. Portanto, um próximo cenário a ser investigado é a implementação da RDNA em *hardware* dedicado, dado que a porta física da NetFPGA que conduz o sinal internamente não é otimizada [Ramanujam 2018]. Ademais, a implementação de uma infraestrutura piloto para continuação do desenvolvimento da arquitetura RDNA se faz necessária. Para isso, pretendemos explorar uma integração da rede RDNA com um orquestrador de nuvem, por exemplo o OpenStack [Corradi et al. 2012].

Referências Bibliográficas

[Abdelsalam et al. 2017] Abdelsalam, A., Clad, F., Filsfils, C., Salsano, S., Siracusano, G., and Veltri, L. (2017). Implementation of virtual network function chaining through segment routing in a linux-based nfv infrastructure. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–5.

[Abramowitz 1974] Abramowitz, M. (1974). *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables,*. Dover Publications, Incorporated.

[Abu-Libdeh et al. 2010] Abu-Libdeh, H., Costa, P., Rowstron, A., O'Shea, G., and Donnelly, A. (2010). Symbiotic routing in future data centers. *SIGCOMM Comput. Commun. Rev.*, 41(4).

[Adrichem et al. 2014] Adrichem, N. L. M. v., Asten, B. J. v., and Kuipers, F. A. (2014). Fast recovery in software-defined networks. In *Proceedings of the 2014 Third European Workshop on Software Defined Networks*, EWSDN '14, pages 61–66, Washington, DC, USA. IEEE Computer Society.

[Agarwal et al. 2014] Agarwal, K., Dixon, C., Rozner, E., and Carter, J. (2014). Shadow macs: Scalable label-switching for commodity ethernet. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 157–162, New York, NY, USA. ACM.

[Akashi et al. 2006] Akashi, O., Fukuda, K., Hirotsu, T., and Sugawara, T. (2006). Policy-based bgp control architecture for autonomous routing management. In *Proceedings of the 2006 SIGCOMM Workshop on Internet Network Management*, INM '06, pages 77–82, New York, NY, USA. ACM.

[Al-Fares et al. 2008] Al-Fares, M., Loukissas, A., and Vahdat, A. (2008). A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 63–74, New York, NY, USA. ACM.

[Al-Fares et al. 2010] Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. (2010). Hedera: Dynamic flow scheduling for data center networks. In

Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, NSDI'10, pages 19–19, Berkeley, CA, USA. USENIX Association.

[Alizadeh et al. 2014] Alizadeh, M., Edsall, T., Dharmapurikar, S., Vaidyanathan, R., Chu, K., Fingerhut, A., Lam, V. T., Matus, F., Pan, R., Yadav, N., and Varghese, G. (2014). Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 503–514, New York, NY, USA. ACM.

[Alves et al. 2011] Alves, J. L. A., Nascimento, L. F. L., and Albini, L. C. P. (2011). Using the redundant residue number system to increase routing dependability on mobile ad hoc networks. Cyber Journals: Journal of Selected Areas in Telecommunications (JSAT).

[Alves Junior 2012] Alves Junior, J. (2012). Um protocolo de roteamento resistente a ataques blackrole sem detecção de nós maliciosos. In *Dissertação (Mestrado)*, Universidade Federal do Paraná, Setor de Ciências Exatas, Departamento de Informática.

[Armbrust et al. 2010] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A view of cloud computing. *Commun. ACM*, 53(4):50–58.

[Awduche and Jabbari 2002] Awduche, D. O. and Jabbari, B. (2002). Internet traffic engineering using multi-protocol label switching (mpls). *Comput. Netw.*, 40(1):111–129.

[Ballardie et al. 1993] Ballardie, T., Francis, P., and Crowcroft, J. (1993). Core based trees (cbt). *SIGCOMM Comput. Commun. Rev.*, 23(4):85–95.

[Barefoot 2018] Barefoot (2018). Barefoot tofino: World's fastest p4-programmable ethernet switch asics. https://barefootnetworks.com/products/brief-tofino/.

[Barham et al. 2003] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 164–177, New York, NY, USA. ACM.

[Barreto 2008] Barreto, F. (2008). *Esquema de caminhos emergenciais rápidos para amenizar perdas de pacotes*. PhD thesis, Engenharia Elétrica e Informática Industrial - Universidade Tecnológica Federal do Paraná (UTFPR), Curitiba/PR, Brasil.

[Bellman 1958] Bellman, R. (1958). On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90.

[Benson et al. 2010] Benson, T., Akella, A., and Maltz, D. A. (2010). Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 267–280, New York, NY, USA. ACM.

[Bilal et al. 2014] Bilal, K., Malik, S. U. R., Khan, S. U., and Zomaya, A. Y. (2014). Trends and challenges in cloud datacenters. *IEEE Cloud Computing*, 1(1):10–20.

[Black 2000] Black, U. (2000). *IP Routing Protocols: RIP, OSPF, BGP, PNNI and Cisco Routing Protocols*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[Boden et al. 1995] Boden, N. J., Cohen, D., Felderman, R. E., Kulawik, A. E., Seitz, C. L., Seizovic, J. N., and king Su, W. (1995). Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15:29–36.

[Bosshart et al. 2014] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

[Bosshart et al. 2013] Bosshart, P., Gibb, G., Kim, H.-S., Varghese, G., McKeown, N., Izzard, M., Mujica, F., and Horowitz, M. (2013). Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. *SIGCOMM Comput. Commun. Rev.*, 43(4):99–110.

[Boyer and Gomide 2012] Boyer, C. and Gomide, E. (2012). *História da matemática*. Edgard Blücher.

[Bradner and McQuaid 1999] Bradner, S. and McQuaid, J. (1999). Benchmarking methodology for network interconnect devices. http://www.ietf.org/rfc/rfc2544.txt.

[Braun and Menth 2014] Braun, W. and Menth, M. (2014). Wildcard compression of inter-domain routing tables for openflow-based software-defined networking. In *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, pages 25–30.

[Buyya and Dastjerdi 2016] Buyya, R. and Dastjerdi, A. V. (2016). *Internet of Things: Principles and Paradigms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.

[Capone et al. 2015] Capone, A., Cascone, C., Nguyen, A. Q. T., and Sansò, B. (2015). Detour planning for fast and reliable failure recovery in sdn with openstate. In *2015 11th International Conference on the Design of Reliable Communication Networks (DRCN)*, pages 25–32.

[Casado et al. 2012] Casado, M., Koponen, T., Shenker, S., and Tootoonchian, A. (2012). Fabric: A retrospective on evolving sdn. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 85–90, New York, NY, USA. ACM.

[Cercós et al. 2015] Cercós, S. S., Ramos, R. M., Eller, A. C. E., Martinello, M., Ribeiro, M. R., Fagertun, A. M., and Monroy, I. T. (2015). Design of a stateless low-latency router architecture for green software-defined networking. In *SPIE OPTO*, pages 93880I–93880I. International Society for Optics and Photonics.

[Chang et al. 2015] Chang, C., Molahosseini, A., Zarandi, A., and Tay, T. (2015). Residue number systems: A new paradigm to datapath optimization for low-power and high-performance digital signal processing applications. *Circuits and Systems Magazine, IEEE*, 15(4):26–44.

[Chen et al. 2018] Chen, R., Zhang, Z., Govindan, V. P., and Wijnands, I. (2018). BGP Link-State extensions for BIER. Internet-Draft draft-ietf-bier-bgp-ls-bier-ext-02, Internet Engineering Task Force. Work in Progress.

[Chokshi et al. 2009] Chokshi, R., Berezowski, K. S., Shrivastava, A., and Piestrak, S. J. (2009). Exploiting residue number system for power-efficient digital signal processing in embedded processors. In *Proceedings of the 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '09, pages 19–28, New York, NY, USA. ACM.

[Cisco 2015] Cisco, C. (2015). Deploying the resilient ethernet protocol (rep) in a converged plantwide ethernet system (cpwe) design guide.

[Congdon et al. 2014] Congdon, P. T., Mohapatra, P., Farrens, M., and Akella, V. (2014). Simultaneously reducing latency and power consumption in openflow switches. *IEEE/ACM Transactions on Networking*, 22(3):1007–1020.

[Cormen et al. 1990] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (1990). *Introduction to Algorithms*. MIT Press.

[Corradi et al. 2012] Corradi, A., Fanelli, M., and Foschini, L. (2012). Vm consolidation: A real case based on openstack cloud. *Future Generation Computer Systems*, pages –.

[Coutinho 2005] Coutinho, S. (2005). *Números inteiros e criptografia RSA*. Série de computação de matemática. IMPA.

[Cowen 2001] Cowen, L. J. (2001). Compact routing with minimum stretch. *J. Algorithms*, 38(1):170–183.

[CPqD 2014] CPqD, F. (2014). Openflow 1.3 software switch. https://github.com/CPqD/ofsoftswitch13.

[da Silva et al. 2015] da Silva, A. S., Smith, P., Mauthe, A., and Schaeffer-Filho, A. (2015). Resilience support in software-defined networking. *Comput. Netw.*, 92(P1):189–207.

[da Silva and Chaves 2004] da Silva, W. and Chaves, L. (2004). Criptografia rsa e o algoritmo chinês do resto. *INFOCOMP*, 2(1):7–9.

[de Lima et al. 2015] de Lima, D. S. A., Guimarães, R. S., Vassoler, G. L., Liberato, A. B., Martinello, M., and Villaca, R. S. (2015). Avaliação do uso do openflow na recuperação de falhas em data centers centrados nos servidores. In *Anais do XIV Workshop em Desempenho de Sistemas Computacionais e de Comunicação*, WPerformance '15, Recife/PE, Brasil. SBC.

[de Souto Filho 2015] de Souto Filho, A. L. (2015). O teorema chinês dos restos. In *Dissertação (Mestrado)*, Universidade Federal do Maranhão, Centro de Ciências Exatas e Tecnologia, Departamento de Matemática.

[Deering et al. 1994] Deering, S., Estrin, D., Farinacci, D., Jacobson, V., Liu, C.-G., and Wei, L. (1994). An architecture for wide-area multicast routing. *SIGCOMM Comput. Commun. Rev.*, 24(4):126–135.

[Dell 2010] Dell (2010). Shared infrastructure: Scale-out advantages and efects on tco. In *Whitepaper*.

[Desmouceaux et al. 2017] Desmouceaux, Y., Pfister, P., Tollet, J., Townsley, M., and Clausen, T. (2017). Srlb: The power of choices in load balancing with segment routing. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2011–2016.

[Desmouceaux et al. 2018] Desmouceaux, Y., Pfister, P., Tollet, J., Townsley, M., and Clausen, T. (2018). 6lb: Scalable and application-aware load balancing with segment routing. *IEEE/ACM Transactions on Networking*, 26(2):819–834.

[Devlin 1998] Devlin, K. (1998). *The Language of Mathematics: Making the Invisible Visible*. W.H. Freeman.

[Dijkstra 1959] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.

[Ding et al. 1996] Ding, C., Pei, D., and Salomaa, A. (1996). *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*. World Scientific Publishing Co., Inc., River Edge, NJ, USA.

[Dominicini et al. 2017] Dominicini, C. K., Vassoler, G. L., Meneses, L. F., Villaca, R. S., Ribeiro, M. R. N., and Martinello, M. (2017). Virtphy: Fully programmable nfv orchestration architecture for edge data centers. *IEEE Transactions on Network and Service Management*, 14(4):817–830.

[Donahue 2011] Donahue, G. A. (2011). *Network Warrior, 2nd Edition*. O'Reilly Media, Farnham, UK, 2nd edition.

[Dua et al. 2014] Dua, R., Raja, A. R., and Kakadia, D. (2014). Virtualization vs containerization to support paas. In *Proceedings of the 2014 IEEE International Conference on Cloud Engineering*, IC2E '14, pages 610–614, Washington, DC, USA. IEEE Computer Society.

[Filsfils et al. 2014] Filsfils, E. C., Previdi, E. S., Systems, I. C., Decraene, B., Litkowski, S., Orange, Shakir, R., and Communications, J. (2014). Segment Routing Architecture. Internet-Draft Segment Routing Architecture draft-ietf-spring-segment-routing-00, Network Working Group. Standards Track.

[Forbes 2016] Forbes (2016). Roundup of internet of things forecasts and market estimates, 2016. https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/.

[Forum 2013] Forum, T. M. E. (2013). Carrier ethernet management information model. Website. https://www.mef.net/Assets/Technical_Specifications/PDF/MEF_7.2.pdf.

[Gallagher et al. 2012] Gallagher, J., Hua, B., Hua, H., and Kodukula, S. (2012). Optimization of network adapter utilization in etherchannel environment. US Patent 8,248,952.

[Garner 1959] Garner, H. L. (1959). The residue number system. *Transactions on Electronic Computers*, pages 140 – 147.

[Ghorbani et al. 2017] Ghorbani, S., Yang, Z., Godfrey, P. B., Ganjali, Y., and Firoozshahian, A. (2017). Drill: Micro load balancing for low-latency data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 225–238, New York, NY, USA. ACM.

[Gill et al. 2011] Gill, P., Jain, N., and Nagappan, N. (2011). Understanding network failures in data centers: Measurement, analysis, and implications. *SIGCOMM Comput. Commun. Rev.*, 41(4):350–361.

[Gimenez et al. 2006] Gimenez, E. J. C., Vieira, R. R., Cardoso, M. J. S., and Ferrari, G. A. (2006). Engenharia de tráfego nas redes mpls: Uma análise comparativa de

seu desempenho em função de suas diferentes implementações. In *World Congress on Computer Science, Engineering and Technology Education (WCCSETE)*, pages 1–5.

[Giorgetti et al. 2017] Giorgetti, A., Sgambelluri, A., Paolucci, F., Sambo, N., Castoldi, P., and Cugini, F. (2017). Bit index explicit replication (bier) multicasting in transport networks. In *2017 International Conference on Optical Network Design and Modeling (ONDM)*, pages 1–5.

[Girolimetto et al. 2017] Girolimetto, M., Paiva, M. H. M., Tessinari, R. S., Pavan, C., and Lima, F. O. (2017). Two deterministic strategies for finding shortest pairs of edge-disjoint paths. In *Spring School on Networks*, Redes para ciudades inteligentes (SSN) 2017, pages 1–6, Pucón, Chile. IEEE CHILECON, IEEEXplore.

[Gomes 2016] Gomes, R. R. (2016). Key for any route (kar): A resilient routing system. In *Dissertação (Mestrado)*, UFES, Universidade Federal do Espírito Santo.

[Gomes et al. 2016] Gomes, R. R., Liberato, A. B., Dominicini, C. K., Ribeiro, M. R. N., and Martinello, M. (2016). Kar: Key-for-any-route, a resilient routing system. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 120–127.

[Gredler and Goralski 2005] Gredler, H. and Goralski, W. (2005). *The Complete IS-IS Routing Protocol*. Springer London.

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

[Greenberg et al. 2009] Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. (2009). Vl2: a scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 51–62, New York, NY, USA. ACM.

[Greenberg et al. 2008b] Greenberg, A., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. (2008b). Towards a next generation data center architecture: Scalability and commoditization. In *Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow*, PRESTO '08, pages 57–62, New York, NY, USA. ACM.

[Grossschadl 2000] Grossschadl, J. (2000). The chinese remainder theorem and its application in a high-speed rsa crypto chip. In *Computer Security Applications, 2000. ACSAC '00. 16th Annual Conference*, pages 384–393.

[Guedes et al. 2012] Guedes, D., Vieira, L., Vieira, M., Rodrigues, H., and Nunes, R. (2012). Redes definidas por software: uma abordagem sistêmica para o desenvolvimento de pesquisas em redes de computadores. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, (30).

[Guo et al. 2009] Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Tian, C., Zhang, Y., and Lu, S. (2009). BCube: a high performance, server-centric network architecture for modular data centers. *SIGCOMM Comput. Commun. Rev.*, 39(4):63–74.

[Guo et al. 2008] Guo, C., Wu, H., Tan, K., Shi, L., Zhang, Y., and Lu, S. (2008). DCell: a scalable and fault-tolerant network structure for data centers. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 75–86, New York, NY, USA. ACM.

[Gupta et al. 2014] Gupta, A., Vanbever, L., Shahbaz, M., Donovan, S. P., Schlinker, B., Feamster, N., Rexford, J., Shenker, S., Clark, R., and Katz-Bassett, E. (2014). Sdx: A software defined internet exchange. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 551–562, New York, NY, USA. ACM.

[Hamada and Nakasato 2005] Hamada, T. and Nakasato, N. (2005). Infiniband trade association - infiniband architecture specification release 1.0. In *in International Conference on Field Programmable Logic and Applications, 2005*, volume 1, pages 366–373.

[Hardesty 2017] Hardesty, L. (2017). Lumina and noviflow seek to disrupt the mpls router market. Website SDX Central. https://www.sdxcentral.com/articles/news/lumina-noviflow-seek-disrupt-mpls-router-market/2017/10/amp/.

[Hari et al. 2015] Hari, A., Lakshman, T. V., and Wilfong, G. (2015). Path switching: Reduced-state flow handling in sdn using path information. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, pages 36:1–36:7, New York, NY, USA. ACM.

[Harju and Kivimaki 2000] Harju, J. and Kivimaki, P. (2000). Co-operation and comparison of diffserv and intserv: performance measurements. In *Proceedings 25th Annual IEEE Conference on Local Computer Networks. LCN 2000*, pages 177–186.

[He et al. 2015] He, K., Rozner, E., Agarwal, K., Felter, W., Carter, J., and Akella, A. (2015). Presto: Edge-based load balancing for fast datacenter networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM 15, pages 465–478, New York, NY, USA. ACM.

[Hosseini et al. 2007] Hosseini, M., Ahmed, D. T., Shirmohammadi, S., and Georganas, N. D. (2007). A survey of application-layer multicast protocols. *IEEE Communications Surveys Tutorials*, 9(3):58–74.

[IEEE 2009] IEEE (2009). Ieee standard for local and metropolitan area networks - virtual bridged local area networks amendment 12: Forwarding and queuing enhancements for time-sensitive streams. *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, pages C1–72.

[Ionescu-Graff et al. 2004] Ionescu-Graff, A. M., Magee, F., Prakash, S., Tang, B., and Zhu, A. (2004). Quantifying the value propositions of mpls evolution; why and when to migrate to a converged mpls core? In *11th International Telecommunications Network Strategy and Planning Symposium. NETWORKS 2004,*, pages 45–50.

[Iren et al. 1999] Iren, S., Amer, P. D., and Conrad, P. T. (1999). The transport layer: Tutorial and survey. *ACM Comput. Surv.*, 31(4):360–404.

[Iselt et al. 2004] Iselt, A., Kirstadter, A., Pardigon, A., and Schwabe, T. (2004). Resilient routing using mpls and ecmp. In *2004 Workshop on High Performance Switching and Routing, 2004. HPSR.*, pages 345–349.

[Islam et al. 2018] Islam, S., Muslim, N., and Atwood, J. W. (2018). A survey on multicasting in software-defined networking. *IEEE Communications Surveys Tutorials*, 20(1):355–387.

[Jeuk et al. 2014] Jeuk, S., Salgueiro, G., and Zhou, S. (2014). Universal cloud classification (ucc) and its evaluation in a data center environment. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 469–474.

[Jia 2014] Jia, W. K. (2014). A scalable multicast source routing architecture for data center networks. *IEEE Journal on Selected Areas in Communications*, 32(1):116–123.

[Jia et al. 2014] Jia, W. K., Chen, C. Y., and Chen, Y. C. (2014). Alex: An arithmetic-based unified unicast and multicast routing for manets. In *2014 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 2114–2119.

[Jokela et al. 2009] Jokela, P., Zahemszky, A., Esteve Rothenberg, C., Arianfar, S., and Nikander, P. (2009). Lipsin: Line speed publish/subscribe inter-networking. *SIGCOMM Comput. Commun. Rev.*, 39(4):195–206.

[Jorge and Gomes 2006] Jorge, L. and Gomes, T. (2006). Survey of recovery schemes in mpls networks. In *2006 International Conference on Dependability of Computer Systems*, pages 110–118.

[Jouet et al. 2015] Jouet, S., Cziva, R., and Pezaros, D. P. (2015). Arbitrary packet matching in openflow. In *16th IEEE International Conference on High Performance Switching and Routing, HPSR 2015, Budapest, Hungary, July 1-4, 2015*, pages 196–201.

[Kempf et al. 2012] Kempf, J., Bellagamba, E., Kern, A., Jocha, D., Takacs, A., and Sköldström, P. (2012). Scalable fault management for openflow. In *2012 IEEE International Conference on Communications (ICC)*, pages 6606–6610.

[Keshav 1997] Keshav, S. (1997). *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Kim et al. 2008] Kim, C., Caesar, M., and Rexford, J. (2008). Floodless in seattle: a scalable ethernet architecture for large enterprises. In Bahl, V., Wetherall, D., Savage, S., and Stoica, I., editors, *SIGCOMM*, pages 3–14. ACM.

[Kim et al. 2011] Kim, C., Caesar, M., and Rexford, J. (2011). Seattle: A scalable ethernet architecture for large enterprises. *ACM Trans. Comput. Syst.*, 29(1):1:1–1:35.

[Kim and Rexford 2007] Kim, C. and Rexford, J. (2007). Revisiting ethernet: plug-and-play made scalable and efficient. In *IEEE LANMAN*.

[Kleinrock and Kamoun 1977] Kleinrock, L. and Kamoun, F. (1977). Hierarchical routing for large networks – performance evaluation and optimization.

[Knuth 1998] Knuth, D. (1998). *The Art of Computer Programming*, volume 1-3. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Kohler et al. 2000] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. (2000). The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297.

[Kompella et al. 2009] Kompella, R. R., Levchenko, K., Snoeren, A. C., and Varghese, G. (2009). Every microsecond counts: Tracking fine-grain latencies with a lossy difference aggregator. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 255–266, New York, NY, USA. ACM.

[Koponen et al. 2014] Koponen, T., Amidon, K., Balland, P., Casado, M., Chanda, A., Fulton, B., Ganichev, I., Gross, J., Gude, N., Ingram, P., Jackson, E., Lambeth, A., Lenglet, R., Li, S.-H., Padmanabhan, A., Pettit, J., Pfaff, B., Ramanathan, R., Shenker, S., Shieh, A., Stribling, J., Thakkar, P., Wendlandt, D., Yip, A., and Zhang, R. (2014). Network virtualization in multi-tenant datacenters. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 203–216, Berkeley, CA, USA. USENIX Association.

[Kotronis et al. 2012] Kotronis, V., Dimitropoulos, X., and Ager, B. (2012). Outsourcing the routing control logic. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks - HotNets-XI*, pages 55–60, New York, New York, USA. ACM Press.

[Kvalbein et al. 2009] Kvalbein, A., Hansen, A. F., Čičic, T., Gjessing, S., and Lysne, O. (2009). Multiple routing configurations for fast ip network recovery. *IEEE/ACM Trans. Netw.*, 17(2):473–486.

[Landau et al. 2011] Landau, A., Ben-Yehuda, M., and Gordon, A. (2011). Splitx: Split guest/hypervisor execution on multi-core. In *Proceedings of the 3rd Conference on I/O Virtualization*, WIOV'11, pages 1–1, Berkeley, CA, USA. USENIX Association.

[Lantz et al. 2010] Lantz, B., Heller, B., and McKeown, N. (2010). A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets '10*, pages 1–6, New York, New York, USA. ACM Press.

[Lee et al. 2004] Lee, S., Yu, Y., Nelakuditi, S., Zhang, Z.-L., and Chuah, C.-N. (2004). Proactive vs reactive approaches to failure resilient routing. In *IEEE INFOCOM 2004*, volume 1, page 186.

[Li et al. 2017] Li, S., Hu, D., Fang, W., Ma, S., Chen, C., Huang, H., and Zhu, Z. (2017). Protocol oblivious forwarding (pof): Software-defined networking with enhanced programmability. *IEEE Network*, 31(2):58–66.

[Li and Freedman 2013] Li, X. and Freedman, M. J. (2013). Scaling ip multicast on datacenter topologies. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 61–72, New York, NY, USA. ACM.

[Liu and Goutte 2014] Liu, Y. and Goutte, R. (2014). Applications of the chinese remainder theorem in signal processing. In *2014 12th International Conference on Signal Processing (ICSP)*, pages 625–630.

[Long et al. 2012] Long, F., Sun, Z., Zhang, Z., Chen, H., and Liao, L. (2012). Research on tcam-based openflow switch platform. In *2012 International Conference on Systems and Informatics (ICSAI2012)*, pages 1218–1221.

[LOUKISSAS 2008] LOUKISSAS, A. (2008). Implementation and simulation of the two-level lookup. In *Dissertação (Mestrado)*, UC, San Diego.

[MacDavid et al. 2017] MacDavid, R., Birkner, R., Rottenstreich, O., Gupta, A., Feamster, N., and Rexford, J. (2017). Concise encoding of flow attributes in sdn switches. In *Proceedings of the Symposium on SDN Research*, SOSR '17, pages 48–60, New York, NY, USA. ACM.

[Mach and Becvar 2017] Mach, P. and Becvar, Z. (2017). Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys Tutorials*, 19(3):1628–1656.

[Mafioletti et al. 2016] Mafioletti, D. R., Liberato, A. B., Villaça, R. d. S., Dominicini, C. K., Martinello, M., and Ribeiro, M. R. N. (2016). Latency measurement as a virtualized network function using metherxis. *SIGCOMM Comput. Commun. Rev.*, 46(4):14–16.

[Mahalingam et al. 2013] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and Wright, C. (2013). VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. Internet Draft.

[Marchese and Mongelli 2012] Marchese, M. and Mongelli, M. (2012). Simple protocol enhancements of rapid spanning tree protocol over ring topologies. *Comput. Netw.*, 56(4):1131–1151.

[Marchetti et al. 2005] Marchetti, M., Padmaraj, M., Chiruvolu, G., Ali, M., and Nair, S. (2005). Traffic engineering in enterprise ethernet with multiple spanning tree regions. *2005 Systems Communications*, 00(undefined):261–266.

[Martinello et al. 2017] Martinello, M., Liberato, A. B., Beldachi, A. F., Kondepu, K., Gomes, R. L., Villaca, R., Ribeiro, M. R. N., Yan, Y., Hugues-Salas, E., and Simeonidou, D. (2017). Programmable residues defined networks for edge data centres. In *2017 13th International Conference on Network and Service Management (CNSM)*, pages 1–9.

[Martinello et al. 2014] Martinello, M., Ribeiro, M. R. N., De Oliveira, R. E. Z., and De Angelis Vitoi, R. (2014). Keyflow: A prototype for evolving SDN toward core network fabrics. *IEEE Network*, 28(2):12–19.

[Martinez et al. 2010] Martinez, F., de A. Moreira, C., Saldanha, N., and Tengan, E. (2010). *Teoria dos números: um passeio com primos e outros números familiares pelo mundo inteiro*. Projeto Euclides. IMPA.

[Martini et al. 2015] Martini, B. et al. (2015). Latency-aware composition of virtual functions in 5G. In *NetSoft 2015*, pages 1–6.

[McKeown et al. 2008] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G. M., Peterson, L. L., Rexford, J., Shenker, S., and Turner, J. S. (2008). Openflow: enabling innovation in campus networks. *Computer Communication Review*, 38(2):69–74.

[Mell and Grance 2011] Mell, P. M. and Grance, T. (2011). Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States.

[MICROSOFT 2013] MICROSOFT (2013). Windows azure platform case studies. Website. http://www.microsoft.com/windowsazure/evidence/.

[Motiwala et al. 2008] Motiwala, M., Elmore, M., Feamster, N., and Vempala, S. (2008). Path splicing. *SIGCOMM Comput. Commun. Rev.*, 38(4):27–38.

[Moy 1994] Moy, J. (1994). Multicast extensions to ospf. RFC 1584, https://tools.ietf.org/html/rfc1584.

[Moy 1998] Moy, J. (1998). Ospf version 2 - rfc 2328 (standard). (request for comments, 2328). updated by rfcs 5709, 6549. Website. http://www.ietf.org/rfc/rfc2328.txt.

[Mudigonda et al. 2011] Mudigonda, J., Yalagandula, P., Mogul, J., Stiekes, B., and Pouffary, Y. (2011). Netlord: a scalable multi-tenant network architecture for virtualized datacenters. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 62–73, New York, NY, USA. ACM.

[Myers et al. 2004] Myers, A., Ng, T. E., and Zhang, H. (2004). Rethinking the service model: Scaling ethernet to a million nodes.

[Nguyen et al. 2011] Nguyen, G. T., Agarwal, R., Liu, J., Caesar, M., Godfrey, P. B., and Shenker, S. (2011). Slick packets. *SIGMETRICS Perform. Eval. Rev.*, 39(1):205–216.

[Ni and McKinley 1993] Ni, L. M. and McKinley, P. K. (1993). A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76.

[Niranjan Mysore et al. 2009] Niranjan Mysore, R., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V., and Vahdat, A. (2009). Portland: A scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.*, 39(4):39–50.

[NSF 2013] NSF (2013). Nsf awards mil lions to fourteen universities for cloud computing research. Website. www.nsf.gov/news.

[Nunez-Yanez et al. 2016] Nunez-Yanez, J. L., Hosseinabady, M., and Beldachi, A. (2016). Energy optimization in commercial fpgas with voltage, frequency and logic scaling. *IEEE Transactions on Computers*, 65(5):1484–1493.

[Oliveira et al. 2018] Oliveira, W. F., Santos, L. S., Martinello, M., and Sampaio, L. N. (2018). Aprovisionamento de qos por rótulos programáveis para redes definidas por resíduos. In *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos (SBRC)*, SBRC 2018, Campos do Jordão/SP. SBRC.

[ONF 2012] ONF, O. N. F. (2012). Openflow switch specification version 1.3.0 (wire protocol 0x04). https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf.

[Oostenbrink et al. 2017] Oostenbrink, J., van Adrichem, N. L. M., and Kuipers, F. A. (2017). Fast failover of multicast sessions in software-defined networks. *CoRR*, abs/1701.08182.

[Open 2018] Open, B. I. (2018). Smart city research and development platform. Website. https://www.bristolisopen.com/.

[OpenDaylight 2013] OpenDaylight (2013). Opendaylight sdn controller platform (oscp):overview. https://wiki.opendaylight.org/view/OpenDaylight_SDN_Controller_Platform_(OSCP):Overview.

[Oran 1990] Oran, D. (1990). RFC 1142: OSI IS-IS Intra-domain Routing Protocol. Technical report, IETF.

[Oueis et al. 2014] Oueis, J., Calvanese-Strinati, E., Domenico, A. D., and Barbarossa, S. (2014). On the impact of backhaul network on distributed cloud computing. In *2014 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 12–17.

[Pang 2007] Pang, T. (2007). Spanning tree bpdu processing method and system facilitating integration of different native vlan configurations. US Patent App. 11/202,802.

[Pansiot 2010] Pansiot, J.-J. (2010). *Multicast Routing on the Internet*. ISTE.

[Pasquini et al. 2011] Pasquini, R., Verdi, F. L., and Magalhães, M. (2011). Integrating servers and networking using an xor-based flat routing mechanism in 3-cube server-centric data centers. In *Anais do XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2011)*, Campo Grande, MS. Sociedade Brasileira de Computação (SBC).

[Perlman 1985] Perlman, R. (1985). An algorithm for distributed computation of a spanningtree in an extended lan. *SIGCOMM Comput. Commun. Rev.*, 15(4):44–53.

[Pfaff et al. 2009] Pfaff, B., Pettit, J., Koponen, T., Amidon, K., Casado, M., and Shenker, S. (2009). Extending Networking into the Virtualization Layer. In *8th ACM Workshop on Hot Topics inNetworks*, volume VIII, page 6. ACM.

[Pfaff et al. 2015] Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., and Casado, M. (2015). The design and implementation of open vswitch. In *12th USENIX Symposium on*

Networked Systems Design and Implementation (NSDI 15), pages 117–130, Oakland, CA. USENIX Association.

[Popa et al. 2010] Popa, L., Ratnasamy, S., Iannaccone, G., Krishnamurthy, A., and Stoica, I. (2010). A cost comparison of datacenter network architectures. In de Oliveira, J. C., Ott, M., Griffin, T. G., and Médard, M., editors, *CoNEXT*, page 16. ACM.

[Ramanujam 2018] Ramanujam, M. (2018). Netfpga sume reference nic. Website. https://github.com/NetFPGA/NetFPGA-SUME-public/wiki/NetFPGA-SUME-Reference-NIC.

[Ramos et al. 2013] Ramos, R., Martinello, M., and Rothenberg, C. (2013). Data center fault tolerant routing and forwarding: An approach based on encoded paths. In *LADC*.

[Ramos 2013] Ramos, R. M. (2013). Roteamento resiliente em redes de data center utilizando openflow: Uma abordagem centrada em caminhos embarcados na origem. In *Dissertação de Mestrado*, PPGI, UFES.

[Ratnasamy et al. 2006] Ratnasamy, S., Ermolinskiy, A., and Shenker, S. (2006). Revisiting ip multicast. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, pages 15–26, New York, NY, USA. ACM.

[Rosen et al. 2001] Rosen, E., Viswanathan, A., and Callon, R. (2001). RFC 3031: Multiprotocol Label Switching Architecture. Technical report, IETF.

[Rothenberg et al. 2012] Rothenberg, C. E., Nascimento, M. R., Salvador, M. R., Corrêa, C. N. A., Cunha de Lucena, S., and Raszuk, R. (2012). Revisiting routing control platforms with the eyes and muscles of software-defined networking. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 13–18, New York, NY, USA. ACM.

[Ryu 2014] Ryu, P. T. (2014). Ryu sdn framework using openflow 1.3. http://osrg.github.io/ryu-book/en/Ryubook.pdf.

[Santana 2013] Santana, G. A. A. (2013). *Data Center Virtualization Fundamentals: Understanding Techniques and Designs for Highly Efficient Data Centers with Cisco Nexus, UCS, MDS, and Beyond*. WebEx Communications, 1st edition.

[Seifert 2000] Seifert, R. (2000). *The Switch Book: The Complete Guide to LAN Switching Technology*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

[Shahbaz et al. 2018] Shahbaz, M., Suresh, L., Feamster, N., Rexford, J., Rottenstreich, O., and Hira, M. (2018). Elmo: Source-Routed Multicast for Cloud Services. *ArXiv e-prints*.

[Sharma and Hellstrand 2003] Sharma, V. and Hellstrand, F. (2003). RFC 3469: Framework for Multi-Protocol Label Switching (MPLS)-based Recovery. Technical report, IETF.

[Shishira and Vipin 2015] Shishira, M. and Vipin, K. (2015). Centralized group key management scheme for secure multicast communication using elgamal. In *International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)*, volume 2, pages 18–21.

[Siddiqi and Nandy 2005] Siddiqi, A. and Nandy, B. (2005). Improving network convergence time and network stability of an ospf-routed ip network. In *Proceedings of the 4th IFIP-TC6 International Conference on Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communication Systems*, NETWORKING'05, pages 469–485, Berlin, Heidelberg. Springer-Verlag.

[Singh et al. 2015] Singh, A., Ong, J., Agarwal, A., Anderson, G., Armistead, A., Bannon, R., Boving, S., Desai, G., Felderman, B., Germano, P., Kanagala, A., Provost, J., Simmons, J., Tanda, E., Wanderer, J., Hölzle, U., Stuart, S., and Vahdat, A. (2015). Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. *SIGCOMM Comput. Commun. Rev.*, 45(4):183–197.

[Singla et al. 2012] Singla, A., Hong, C.-Y., Popa, L., and Godfrey, P. B. (2012). Jellyfish: Networking data centers randomly. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 17–17, Berkeley, CA, USA. USENIX Association.

[Sivaraman et al. 2015] Sivaraman, A., Kim, C., Krishnamoorthy, R., Dixit, A., and Budiu, M. (2015). Dc.p4: Programming the forwarding plane of a data-center switch. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 2:1–2:8, New York, NY, USA. ACM.

[Spalla et al. 2014] Spalla, E., Mafioletti, D. R., Liberato, A., Martinello, M., and Villaça, R. (2014). Resiliência no plano de controle para redes definidas por software. In *Anais do V Workshop de Pesquisa Experimental em Internet do Futuro*, WPEIF 2014, pages 1–4, Florianópolis/SC. XXXII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC).

[Spalla et al. 2015] Spalla, E., Mafioletti, D. R., Liberato, A., Rothenberg, C. E., Camargos, L., Martinello, M., and Villaça, R. (2015). Estrategias para resiliencia em sdn: Uma abordagem centrada em multi-controladores ativamente replicados. In *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos (SBRC)*, SBRC 2015, Vitória/ES. SBRC.

[Spalla et al. 2016] Spalla, E. S., Mafioletti, D. R., Liberato, A. B., Ewald, G., Rothenberg, C. E., Camargos, L., Villaca, R. S., and Martinello, M. (2016). Ar2c2: Actively replicated controllers for sdn resilient control plane. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 189–196.

[Staff 2013] Staff, V. (2013). Post vmware staff. Website. www.vmware.com.

[Stephens et al. 2012] Stephens, B., Cox, A., Felter, W., Dixon, C., and Carter, J. (2012). Past: Scalable ethernet for data centers. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pages 49–60, New York, NY, USA. ACM.

[Sun et al. 2015] Sun, P., Yu, M., Freedman, M. J., Rexford, J., and Walker, D. (2015). Hone: Joint host-network traffic management in software-defined networks. *J. Netw. Syst. Manage.*, 23(2):374–399.

[Suurballe and Tarjan 1984] Suurballe, J. W. and Tarjan, R. E. (1984). A quick method for finding shortest pairs of disjoint paths. *Networks*, 14(2):325–336.

[Swallow et al. 2005] Swallow, G., Pan, P., and Atlas, A. (2005). RSVP-TE fast reroute. RFC 4090, http://www.ietf.org/rfc/rfc4090.txt.

[Testa and Pavesi 2017] Testa, F. and Pavesi, L. (2017). *Optical Switching in Next Generation Data Centers*. Springer Publishing Company, Incorporated, 1st edition.

[Thorup and Zwick 2001] Thorup, M. and Zwick, U. (2001). Compact routing schemes. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '01, pages 1–10, New York, NY, USA. ACM.

[Trois et al. 2016] Trois, C., Fabro, M. D. D., de Bona, L. C. E., and Martinello, M. (2016). A survey on sdn programming languages: Toward a taxonomy. *IEEE Communications Surveys Tutorials*, 18(4):2687–2712.

[van Asten et al. 2014] van Asten, B. J., van Adrichem, N. L. M., and Kuipers, F. A. (2014). Scalability and resilience of software-defined networking: An overview. *CoRR*, abs/1408.6760.

[Vasseur et al. 2004] Vasseur, J.-P., Pickavet, M., and Demeester, P. (2004). Network recovery: Protection and restoration of optical, sonet-sdh, ip, and mpls.

[Vassoler et al. 2014] Vassoler, G., Paiva, M., Ribeiro, M., and Segatto, M. (2014). Twin datacenter interconnection topology. *Micro, IEEE*, 34(5):8–17.

[Vassoler 2015] Vassoler, G. L. (2015). *TRIIIAD: Uma Arquitetura para Orquestração Automônica de Redes de Data Center Centrado em Servidor.* PhD thesis, Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal do Espírito Santo (PPGEE/UFES), Vitória/ES, Brasil.

[Verdi et al. 2010] Verdi, F. L., Rothenberg, C. E., Pasquini, R., and Magalhães, M. (2010). Novas arquiteturas de data center para cloud computing. *Minicursos do XXVIII - Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, pages 103–152.

[vSwitch 2014] vSwitch, O. (2014). An Open Virtual Switch. http://openvswitch.org/.

[Waitzman et al. 1988] Waitzman, D., Partridge, C., and Deering, S. (1988). Distance vector multicast routing protocol. RFC 1075, https://tools.ietf.org/html/rfc1075.

[Wang and Crowcroft 1992] Wang, Z. and Crowcroft, J. (1992). Analysis of shortest-path routing algorithms in a dynamic network environment. *SIGCOMM Comput. Commun. Rev.*, 22(2):63–71.

[Wollschlaeger et al. 2017] Wollschlaeger, M., Sauter, T., and Jasperneite, J. (2017). The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. *IEEE Industrial Electronics Magazine*, 11(1):17–27.

[Wu et al. 2009] Wu, H., Lu, G., Li, D., Guo, C., and Zhang, Y. (2009). Mdcube: A high performance network structure for modular data center interconnection. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '09, pages 25–36, New York, NY, USA. ACM.

[Xia et al. 2015] Xia, W., Wen, Y., Foh, C. H., Niyato, D., and Xie, H. (2015). A survey on software-defined networking. *IEEE Communications Surveys Tutorials*, 17(1):27–51.

[Yamanaka et al. 2014] Yamanaka, H., Kawai, E., Ishii, S., and Shimojo, S. (2014). Openflow networks with limited l2 functionality. *Thirteenth International Conference on Networks (ICN 2014)*, pages 221–229.

[Yang and Wetherall 2006] Yang, X. and Wetherall, D. (2006). Source selectable path diversity via routing deflections. *SIGCOMM Comput. Commun. Rev.*, 36(4):159–170.

[Yuan et al. 2007] Yuan, X., Nienaber, W., Duan, Z., and Melhem, R. (2007). Oblivious routing for fat-tree based system area networks with uncertain traffic demands. *SIGMETRICS Perform. Eval. Rev.*, 35(1):337–348.

[Yuan et al. 2009] Yuan, X., Nienaber, W., Duan, Z., and Melhem, R. (2009). Oblivious routing in fat-tree based system area networks with uncertain traffic demands. *IEEE/ACM Trans. Netw.*, 17(5):1439–1452.