AGREGAÇÃO DINÂMICA DE ENLACES EM AMBIENTES DE REDES DEFINIDAS POR SOFTWARE

RONALDO RESENDE ROCHA JUNIOR

AGREGAÇÃO DINÂMICA DE ENLACES EM AMBIENTES DE REDES DEFINIDAS POR SOFTWARE

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Marcos Augusto Menezes Vieira
Coorientador: Antônio Alfredo Ferreira Loureiro

Belo Horizonte

Dezembro de 2017

Rocha Junior, Ronaldo Resende

R672a Agregação dinâmica de enlaces em ambientes de redes definidas por software / Ronaldo Resende Rocha Junior. — Belo Horizonte, 2017
xxii, 42 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais — Departamento de Ciência da Computação
Orientador: Marcos Augusto Menezes Vieira
Coorientador: Antônio Alfredo Ferreira Loureiro

1. Computação - Teses. 2. Redes de Computadores. 3. Arquitetura de Redes de Computadores I. Orientador. II. Coorientador. III. Título.

CDU 519.6*22(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Agregação dinâmica de links em ambientes de redes definidas por software

RONALDO RESENDE ROCHA JUNIOR

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. MARCOS AUGUSTO MENEZES VIEIRA - Orientador
Departamento de Ciência da Computação - UFMG

PROF. ANTONIO ALFREDO FERREIRA LOUREIRO
Departamento de Ciência da Computação - UFMG

PROF. ÍTALO FERNANDO SCOTÁ CUNHA
Departamento de Ciência da Computação - UFMG

PROFA. RAQUEL APARECIDA DE FREITAS MINI
Departamento de Ciência da Computação - PUC-MG

Belo Horizonte, 27 de Dezembro de 2017.

Dedico este trabalho ao meu querido avô e eterno exemplo, Sebastião.

Agradecimentos

Em primeiro lugar, agradeço à minha querida esposa por todo apoio, paciência e inspiração durante o processo desta pesquisa. Obrigado à minha mãe por sempre acreditar e investir em minha carreira profissional e acadêmica no mundo de TI. Agradeço à minha avó, aos meus tios e à minha prima Laura pelo suporte ao longo dos anos. Obrigado também aos meus amigos próximos pela torcida durante esta longa batalha. Agradeço, por fim, à CAPES e a FAPEMIG pelas bolsas de estudos providas, e ao meu orientador e co-orientador pela confiança em meu trabalho.

"There's No Place Like 127.0.0.1"
(Autor desconhecido)

Resumo

A técnica de agregação de enlaces provê robustez às redes de computadores, além de ser uma solução para o problema de saturação de enlaces. Essa técnica combina várias interfaces físicas para criar um enlace virtual, somando as respectivas bandas existentes. Além de aumentar a vazão para a transmissão de dados, tal técnica provê uma recuperação rápida e transparente caso um determinado enlace fique indisponível. Visto que a utilização das Redes Definidas por Software (SDN) em ambientes empresariais aumenta a cada dia, este trabalho aprensenta uma forma automática para a criação das agregações de enlaces em tais ambientes. Isso traz uma maior disponibilidade de serviços, entre outros benefícios.

Para tanto, uma arquitetura que permite agregar enlaces de forma automática, escalável e auto-adaptável foi definida e implementada. Como forma de avaliar esta implementação, três algoritmos foram criados utilizando diferentes premissas de agregação de enlaces: Tabela *Hash*, Análise de Tráfego e *Round-Robin* Virtual. Todas as implementações foram testadas em ambientes virtuais e reais. Em ambos, a plataforma aberta *Open vSwitch* foi utilizada para a comutação de pacotes e o controlador *Ryu* foi a escolhido para controlar os *switches*.

Palavras-chave: Agregação de enlaces, Redes Definidas por Software, *Open vSwitch*, Redes Programáveis, LACP.

Abstract

The link aggregation technique not only provides robustness to computer networks, but can also be a solution to the link saturation problem. This technique combines several physical interfaces to create a virtual link, adding up their existing bandwidth. In addition to increasing the throughput for data transmission, such a technique provides a quick and transparent recovery if a particular link becomes unavailable. Considering that the use of Software-Defined Networking (SDN) in business environments increases every day, this research presents a way to create link aggregations in such environments. This brings a better availability of services, among other benefits.

To do so, an architecture that allows automatic, scalable and self-adaptive link aggregations was defined and implemented. As a way of evaluating such implementation, three algorithms were created using different premises: Hash Table, Traffic Analysis and Virtual Round-Robin. All implementations were tested in virtual and real environments. In both, the Open vSwitch open platform was used for packet switching and the Ryu controller was chosen to control the switches.

Keywords: Link Aggregation, Software-Defined Networking, Open vSwitch, Programmable Networks, LACP.

Lista de Figuras

2.1 LAN x WAN 6
2.2 Planos existentes em equipamentos de redes (adaptado de [Morreale & Anderson, 2014]) 7
2.3 Componentes básicos em redes SDN 8

4.1 Arquitetura proposta neste trabalho 14
4.2 Exemplo da estrutura de dados das agregações identificadas 17
4.3 Exemplo básico de um *broadcast storm* 18
4.4 Código do mecanismo de prevenção de *loop* 18

6.1 Topologia virtual com agregação de enlaces 24
6.2 Comparativo entre tráfegos TCPxUDP entre os três algoritmos implementados 25
6.3 Utilização das interfaces durante a queda da interface 01 27
6.4 Topologia utilizada para provar a teoria do intra-fluxo 28
6.5 Imagem da coleta de tráfego realizada entre as interfaces agregadas 29
6.6 Tela principal do projeto LEDE 31
6.7 Laboratório real com agregação de enlaces 32
6.8 Equipamentos utilizados no laboratório real 32
6.9 Função de Distribuição Cumulativa para os resultados de laboratório virtual (Tráfego artificial) 33
6.10 Função de Distribuição Cumulativa para os resultados de laboratório virtual (Tráfego real) 34
6.11 Função de Distribuição Cumulativa para os resultados da vida real 35
6.12 Gráfico de dispersão dos índices de justiça obtidas nos testes 35

Lista de Tabelas

6.1 Tráfego gerado artificialmente: banda total medida por cada estação transmissora (em Mbps) 25
6.2 Tráfego realista: banda total medido por cada porta (em Mbps) 26
6.3 Vazão total medida para cada transmissão de cada estação(em Mbps) . . . 32

Sumário

Agradecimentos **ix**

Resumo **xiii**

Abstract **xv**

Lista de Figuras **xvii**

Lista de Tabelas **xix**

1 Introdução **1**

2 Fundamentação Teórica **5**

3 Trabalhos Relacionados **9**

3.1 LACP *(Link Aggregation Control Protocol)* 9

3.2 Outras abordagens 10

4 Metodologia **13**

4.1 Arquitetura 13

4.2 Implementação 14

4.2.1 Principais eventos processados pelo controlador *Ryu* 15

4.3 Identificação das agregações de enlaces 16

4.4 Mecanismo de prevenção de *loop* 17

4.5 Método de prevenção de falhas 18

4.6 Tráfego intra-fluxo 18

5 Algoritmos de agregação de enlace **19**

5.1 Tabela *Hash* 20

5.2 Análise de Tráfego 20

5.3 *Round-Robin* virtual 21

6 Avaliação Experimental e Resultados **23**

6.1 Topologia virtual 24

6.1.1 Ambiente virtual 24

6.1.2 Banda total utilizada 24

6.1.3 Tráfego realista 26

6.1.4 Queda das interfaces 26

6.1.5 Tráfego intra-fluxo 27

6.2 Experimento com laboratório real 29

6.2.1 LEDE 30

6.2.2 Tráfego real 30

6.3 Discussão 33

7 Conclusão e Trabalhos Futuros **37**

Referências Bibliográficas **39**

Capítulo 1

Introdução

A gerência de redes de computadores é um grande desafio pois elas se tornaram sistemas complexos [Ranum et al., 1998]. Os equipamentos de redes atuais, tais como roteadores, *firewalls* e *switches*, são considerados verdadeiras caixas pretas [Moore & Nettles, 2001a]. Cada fabricante é responsável por criar os sistemas operacionais que rodam em seus equipamentos de redes e, mesmo que os algoritmos de roteamento sejam os mesmos, a forma como eles são implementados pode ser bastante diferente, podendo, inclusive, levar a problemas de interoperabilidade. Para cada equipamento de rede instalado em um site, os administradores precisam realizar configurações únicas a fim de garantir o seu pleno funcionamento [Tennenhouse & Wetherall, 2007].

O problema do gargalo (*bottleneck*) no tráfego é um dos principais desafios existentes em ambientes de redes [Carter & Crovella, 1996]. Isso acontece, por exemplo, quando existem vários usuários que utilizam o mesmo servidor, o que pode saturar o enlace que conecta o *switch* àquele servidor. Esse é um dos motivos pelos quais muitas vezes avaliações incorretas são feitas a respeito da lentidão em ambientes de rede, levando à troca ou *upgrade* desnecessários de equipamentos. É importante otimizar recursos, incluindo a banda utilizada em ambientes de *data centers*. Além disso, é necessário criar ambientes que possuem topologias dinâmicas, altamente configuráveis e gerenciáveis de forma autônoma.

A agregação de enlaces permite ter uma rede com maior disponibilidade, tolerante à falhas e com maior potencial de vazão, o que é ideal em ambientes corporativos [Floyd et al., 1995]. Nos *switches* clássicos, essa técnica é realizada através do protocolo aberto LACP (*Link Aggregation Control Protocol*) ou os diversos protocolos proprietários dos fabricantes (Cisco: *Port Aggregation Protocol*, Huawei: *Eth-Trunk*, Juniper: *Aggregated Ethernet*, entre outros). Porém, o LACP possui várias limitações. A configuração é feita de forma manual, o número máximo de enlaces físicos agregados é limitado a

oito e todas as interfaces de rede devem operar com a mesma velocidade para serem agregadas.

Entretanto, a aplicação da técnica de agregação de enlaces ainda é pouco estudada em redes SDN. Atualmente, a única forma de realizar essa prática é utilizando o protocolo LACP em conjunto com a plataforma *Open vSwitch*, mas esse cenário possui restrições. Diante disso, o objetivo do presente trabalho é desenvolver um meio de realizar agregação de enlaces em ambientes SDN com a finalidade de aumentar a vazão de dados de uma rede. Para alcançar essa meta, a estrutura de dados do protocolo *OpenFlow* é utilizada, fazendo com que o controlador tenha acesso direto e facilitado aos pacotes (e seus cabeçalhos) que trafegam na rede. Assim, a agregação pode ser realizada de forma dinâmica e sem a necessidade da configuração manual por parte de um administrador de redes.

Para resolver esses problemas, é proposto um sistema com agregação de enlaces em ambientes SDN. Sua implementação é feita diretamente no controlador SDN e três premissas foram utilizadas para avaliar o desempenho da solução. Na primeira, um *hash* é criado utilizando os parâmetros do pacote enviado (endereços físicos e IPs) de uma estação de origem para uma estação de destino para determinar qual interface da agregação será utilizada para o fluxo. Na segunda, o tráfego das interfaces agregadas é medido e a interface com menor utilização é selecionada para transmitir o próximo fluxo. Na última, regras utilizando todas as interfaces são criadas com pesos diferentes, que são alternados dentro de um período de tempo (a cada 500 milissegundos, por exemplo), fazendo com que um mesmo fluxo transite por diferentes interfaces agregadas.

Os algoritmos de agregação podem ser classificados em inter-fluxo e intra-fluxo. O tipo inter-fluxo ocorre quando cada fluxo é separado em caminhos distintos, mas não existe divisão do fluxo. O tipo intra-fluxo ocorre quando o fluxo é dividido em vários subfluxos com caminhos distintos.

Com o padrão *OpenFlow* não é possível acessar todos os campos do pacote, por exemplo, o número de sequência do TCP. Além disso, o *Open vSwitch* não possui funcionalidades para fazer a agregação intra-fluxo. Com isso, foi projetado e construído um novo *switch* que permite fazer tal tipo de agregação.

As contribuições deste trabalho são:

- criação de um sistema que permite utilizar a técnica de agregação de enlaces intra e inter-fluxo em ambientes SDN;

- proposta de três algoritmos para a agregação;

- avaliação dos três algoritmos em ambiente virtual e real, incluindo comparações com o LCAP e a utilização nativa do *Linux Bounding*;
- projeto e implementação de um novo *switch* que consegue fazer a agregação intra-fluxo.

Para validar o funcionamento e desempenho das premissas criadas, as ferramentas *iperf* e *tcpreplay* foram utilizadas para a geração de tráfego. Além disso, dois ambientes distintos foram criados: uma máquina virtual simulando uma rede de computadores com *switches* e estações comunicando entre si, e um laboratório real com dois roteadores da marca *TP-Link* rodando um *firmware* customizado. Com isso, o sistema proposto possui várias vantagens sobre o estado da arte:

- **Independente de protocolo**: A solução não depende de outros protocolos além do *OpenFlow*. Isso significa que não existe a necessidade de nenhum conhecimento ou configuração especial a ser realizada no *switch* ou no controlador. A agregação é detectada, modificada e removida de forma automática pelo sistema.
- **Distribuição de banda justa**: Utilizando diferentes estratégias para a tomada de decisão ao criar os fluxos de transmissão de pacotes, as interfaces da agregação podem ter um balanceamento de tráfego uniforme.
- **Agregação autônoma**: A agregação pode ser criada, modificada ou removida automaticamente pelo controlador sem a necessidade de intervenção do administrador de redes.
- **Código aberto**: Utilizando a estrutura de dados e tabelas presentes no *OpenFlow*, o fluxo de transmissão de pacotes entre os enlaces de uma agregação é gerenciado pelo controlador SDN. Isso faz com que o uso de protocolos de agregação proprietários não seja mais necessário.
- **Aumento da quantidade de interfaces agregadas**: Uma das limitações do protocolo LACP é a utilização máxima de oito interfaces para a agregação. Ao utilizar o LACP na plataforma *Open vSwitch*, esse valor cai para quatro portas (uma limitação referenciada na documentação da plataforma). Ao migrar a gerência da agregação desses enlaces para o controlador SDN, essa limitação deixa de existir.
- **Velocidade de transmissão entre enlaces**: Outra limitação do protocolo LACP é a necessidade de utilizar interfaces com a mesma configuração de velocidade na agregação, ou seja, interfaces com velocidade 100 Mbps não podem ser

agregadas com interfaces com velocidade de 1 Gbps, por exemplo. Essa limitação existe por motivos de implementações do protocolo, na forma como ele realiza e mantém a agregação configurada. Tal limitação não existiria em ambientes SDN.

Este trabalho está estruturado da seguinte forma: no capítulo 2, alguns fundamentos teóricos são discutidos. Alguns trabalhos anteriores relacionados às técnicas utilizadas nesta pesquisa são discutidos no capítulo 3. No capítulo 4, a metodologia é descrita com mais detalhes. O capitulo 5 explica como cada algoritmo implementado funciona, enquanto o capítulo 6 apresenta a avaliação experimental e os resultados obtidos. Por fim, o capítulo 7 apresenta a conclusão e propõe trabalhos futuros.

Capítulo 2

Fundamentação Teórica

O modelo clássico da rede de computadores foi criado na década de 1970 após diversas pesquisas, resultando na famosa ARPANET [McQuillan et al., 1980]. Naquele momento, a utilização de um esquema de endereçamento com apenas 32 bits (resultando um total de 2^{32} endereços únicos) parecia suficiente. Além disso, quando a internet surgiu ela era considerada algo estático, uma rede onde a topologia não deveria mudar no decorrer do tempo. Entretanto, desde 2011 não existem mais endereços IPv4 disponíveis, forçando a adoção de um novo modelo de endereçamento, o IPv6, que disponibiliza uma quantidade virtual ilimitada de endereços. A topologia se tornou dinâmica em decorrência dos protocolos de roteamento utilizados, sejam para a comutação de pacotes dentro das redes locais ou entre as redes locais.

As redes modernas de computadores evoluíram para algo complexo e por isso, a sua administração é um grande desafio tanto para profissionais quanto para pesquisadores [Ranum et al., 1998].

Em um contexto geral, podemos dividir as redes em duas categorias principais representadas pela figura 2.1:

- **Local Area Networks**: rede local ou LAN, que interliga computadores e dispositivos dentro de um mesmo espaço físico chamados de sites. É comum em empresas, residências ou universidades possibilitando a troca de dados entre todos os dispositivos participantes.

- **Wide Area Network**: rede de longa distância ou WAN, capaz de abranger cidades ou até mesmo um continente. Duas ou mais LANs conectadas formam uma WAN, e como a internet pode ser considerada um conjunto de LANs interconectadas, o roteador local conecta a LAN de uma residência à Internet (WAN).

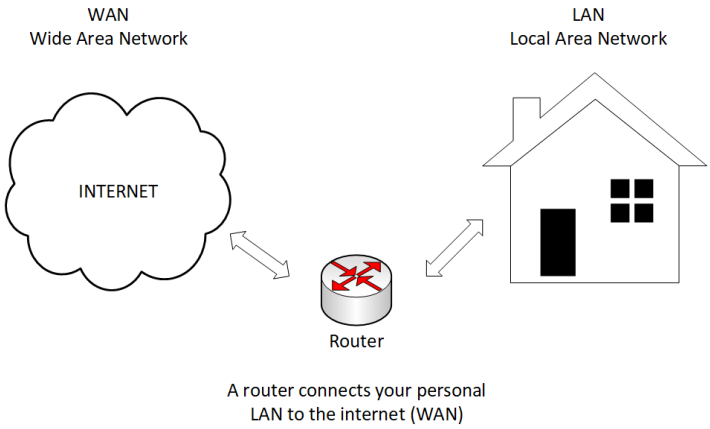

Figura 2.1. LAN x WAN

Os equipamentos de redes possuem três planos distintos para realizar a comutação de pacotes, representados pela figura 2.2:

- **Plano de Dados**: responsável por controlar a comutação de pacotes recebidos nas portas físicas do equipamento. Além disso, controla a modificação de cabeçalho, o encaminhamento, o buffer e o agendamento de envio dos pacotes. Quando um pacote é recebido, o seu cabeçalho é analisado e o plano de controle utiliza uma tabela interna de encaminhamento para tomar a decisão de descartá-lo ou encaminhá-lo. Caso ele necessite ser encaminhado, uma porta de destino é escolhida. Todo esse processo deve ser realizado com muita agilidade para evitar congestionamentos ou atrasos.

- **Plano de Controle**: responsável por configurar e atualizar as informações da tabela de encaminhamento, permitindo que o plano de dados tome suas decisões sem a necessidade de consultas ao plano de controle. Também gerencia diversos protocolos de controle que afetam diretamente a tabela de encaminhamento de acordo com as configurações existentes ou do tipo de equipamento utilizado. Este plano possui uma memória para processamento dos dados e uma memória para armazenamento das configurações (tabela de roteamento, por exemplo).

- **Plano de Gerenciamento**: responsável pela configuração e monitoramento dos equipamentos. É neste plano que o sistema operacional roda, aplicando as configurações diretamente nos planos de controle e de dados, além de obter informações e estatísticas em tempo real (tráfego de portas, quantidade de pacotes trafegados, etc) do equipamento.

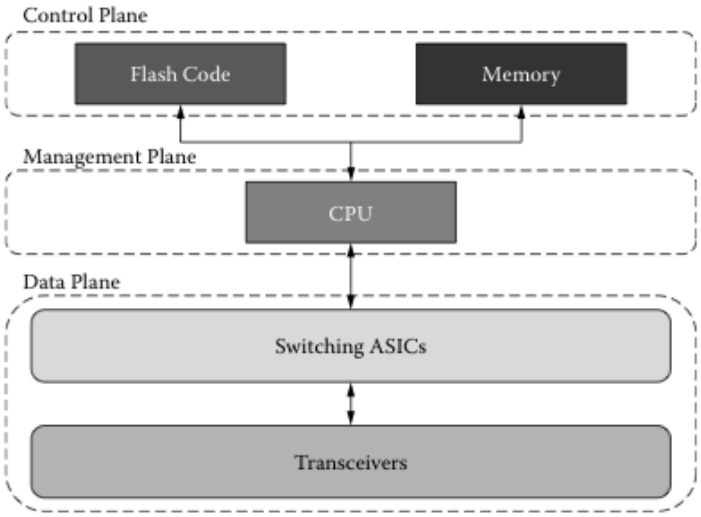

Figura 2.2. Planos existentes em equipamentos de redes (adaptado de [Morreale & Anderson, 2014])

Para cada equipamento de rede instalado em um site, os administradores precisam realizar configurações únicas a fim de garantir o seu pleno funcionamento [Tennenhouse & Wetherall, 2007]. Em 2008, pesquisadores iniciaram dois projetos com o objetivo comum de criar um novo modelo de gerenciamento de redes [McKeown et al., 2008; Gude et al., 2008]. Um grupo da startup Nicira criou um sistema operacional de redes chamado NOX, e outro grupo da mesma empresa em conjunto com pesquisadores da universidade de Stanford criou uma interface para controle de *switches* chamado *OpenFlow*. Surgia assim a Rede Definida por Software, ou *Software-Defined Networking* (SDN), que teve seu padrão publicado em 2011 pela organização Open Networking Foundation. O objetivo dessa organização é promover a utilização e adoção do SDN através de um padrão aberto, e empresas como Cisco, Juniper, Broadcom, Google, Microsoft, Yahoo e Dell são alguns dos seus vários membros.

As Redes Definidas por Software possuem como objetivo primário separar o plano de controle do plano de dados [Kim et al., 2013]. Essa separação já existe fisicamente em equipamentos de redes mais modernos, mas ela é contida fisicamente em um mesmo aparato, fazendo com que os planos continuem atrelados ao mesmo equipamento de um único fabricante. As redes SDN propõem uma separação total entre os dois planos. Dessa forma, os administradores de redes definem, através de configurações, como o plano de controle irá definir as regras de roteamento de pacotes que o plano de dados irá executar. Isso faz com que o plano de dados apenas encaminhe os pacotes utilizando regras definidas anteriormente, sem fazer parte do processo de decisão de roteamento. A partir dessa configuração, vários equipamentos podem ser gerenciados por apenas

um plano de controle, facilitando as configurações lógicas em *data centers* e diminuindo a complexidade de monitoramento do tráfego.

Existem três componentes básicos em redes SDN, representados pela figura 2.3:

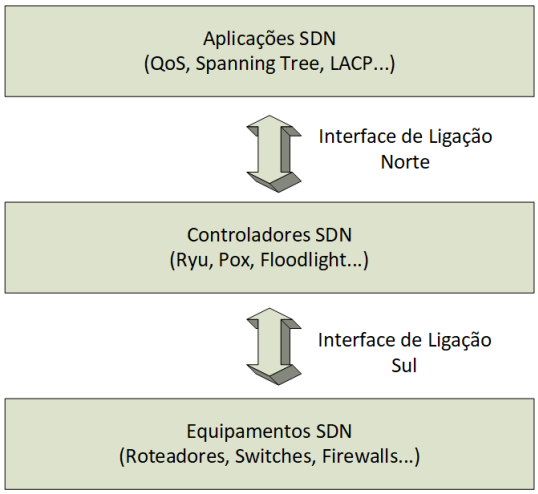

Figura 2.3. Componentes básicos em redes SDN

- **Aplicações SDN**: interface de configuração que os administradores de redes utilizam para configurar as redes. Através delas, os administradores podem definir os fluxos dos pacotes podem ser definidos, o redirecionamento de tráfego para fins de inspeções ou até mesmo a reconfiguração dinâmica dos equipamentos em casos de falhas.

- **Controladores SDN**: responsável por gerenciar a rede local. Além de manter um mapa global da rede, realiza as definições das políticas e decisões de roteamento, encaminhamento, redirecionamento e balanceamento de carga. Uma rede pode possuir um ou mais controladores. A utilização de diversos controladores pode trazer benefícios como segurança, redundância e proteção à falhas.

- **Equipamentos SDN**: responsável por comutar pacotes nas redes e possui três componentes. A API é a interface de comunicação com o controlador e é responsável por executar as modificações locais na tabela de roteamento, além de fornecer estatísticas e informações em tempo real. A camada de abstração é responsável por armazenar uma ou mais tabelas de encaminhamento de pacotes, que são constantemente atualizadas de acordo com as configurações existentes no controlador. Por fim, a camada de processamento de pacotes realiza o envio de pacotes através das interfaces físicas de acordo com as regras da camada de abstração.

Capítulo 3

Trabalhos Relacionados

Alguns trabalhos foram cruciais para a fundamentação teórica desta dissertação. Em [Macedo et al., 2015] as mais recentes pesquisas na área de redes SDN são discutidas e expostas. Os autores [Kim & Feamster, 2013] também destacam as diferenças e benefícios entre o modelo clássico e o SDN. No minicurso [Guedes et al., 2012] o funcionamento de ferramentas como *Mininet* e *iperf* são explorados e exemplificados. O trabalho realizado por [Carter & Crovella, 1996], apesar de ser relativamente antigo, elucidou os reais problemas do gargalo de tráfego nas redes de computadores. Por fim, os trabalhos de [Moore & Nettles, 2001b] e [Tennenhouse & Wetherall, 2002] também foram importantes durante o processo inicial de pesquisa.

3.1 LACP *(Link Aggregation Control Protocol)*

O protocolo aberto LACP (*Link Aggregation Control Protocol*) permite a implantação da agregação de enlaces nos *switches* clássicos. Esse protocolo foi publicado em 1997 por [Seaman, 1999] e sua última versão é de 2014, com o padrão 802.1AX-2014 [IEEE, 2014]. O LACP é amplamente utilizado em ambientes clássicos de redes, mas a sua aplicação em redes SDN ainda não é muito pesquisada.

Pra criar o fluxo, o LACP analisa vários parâmetros, tais como endereços MAC, IP e portas de origem e de destino, porta e etiqueta MPLS. A partir do *hash* desses parâmetros, o protocolo escolhe a porta para o fluxo, fazendo com que todos os pacotes deste fluxo passem somente nessa interface. Quando surge um novo fluxo, um novo *hash* é calculado e, provavelmente, outra interface será utilizada para ele. Assim, o LACP usa todas as interfaces da agregação, espalhando os fluxos entre elas.

Para criar uma agregação de enlaces utilizando o protocolo LACP, o administrador de redes deve realizar a configuração de forma manual. Ao ser habilitado, o

protocolo cria uma interface virtual e a quantidade de enlaces físicos não pode ser maior do que oito e todas as interfaces do *switch* devem estar configuradas com a mesma velocidade. Não existe uma explicação clara para o motivo da limitação da quantidade de portas, porém acredita-se que seja por motivos de escalabilidade.

O protocolo LACP pode ser empregado em redes SDN através da plataforma *Open vSwitch*. Por ser basicamente uma máquina virtual, essa plataforma geralmente é instalada em estações de trabalho ou servidores, impossibilitando sua utilização em equipamentos SDN físicos.

3.2 Outras abordagens

A plataforma *Open vSwitch* foi criada para ser um *switch* virtual capaz de executar em qualquer ambiente [Pfaff et al., 2009]. Pelo fato de ser bastante completa e robusta, logo foi incorporada nas redes SDN. Entretanto, considerando a técnica de agregação de enlaces, essa plataforma não é eficaz. A quantidade máxima de agregação de portas físicas é bastante limitada, não podendo ultrapassar quatro unidades (uma limitação da própria plataforma). Portanto, a solução não é escalável, sendo ineficiente em ambientes como *data centers*, que possuem centenas de equipamentos.

No trabalho de [Jouet & Pezaros, 2017], os autores apresentam uma arquitetura livre de plataformas, protocolos ou linguagens de programação para monitorar e controlar equipamentos de redes. Inspirado por este trabalho, foram realizados testes com transmissões intra-fluxos que são discutidas ao decorrer deste trabalho.

No sistema operacional Linux, existe uma biblioteca nativa para agregação de enlaces [Davis et al., 2011]. O comportamento das interfaces agregadas depende do método selecionado, mas os principais são conhecidos como TCP balanceado e SLB balanceado. O primeiro utiliza informações dos pacotes das camadas 2, 3 e 4 para balancear o tráfego e depende que a outra ponta (geralmente um *switch*) esteja com o LACP configurado. Já o segundo utiliza apenas informações da camada 2 e não é dependente do protocolo LACP.

Em [Steinbacher & Bredel, 2015], os autores propõem a implementação do protocolo LACP no Floodlight, um controlador aberto escrito em linguagem Java. Esse trabalho apenas discute a implementação teórica do protocolo, sendo que a configuração do código do protocolo em si não é elaborada.

Já os autores de [Bredel et al., 2014], [Ahn et al., 2009], [Al-Fares et al., 2010] e [Alizadeh et al., 2010] consideram o caminho de um tráfego TCP entre múltiplos enlaces, sem considerar uma possível agregação local deles. Porém, em todos eles são

considerados caminhos através de múltiplos saltos. A diferença deste trabalho para os demais é a abordagem na camada 2, ou seja, a comunicação entre dois *switches* com enlaces agregados entre eles.

O trabalho realizado por [Al-Shabibi et al., 2014] propõe a implementação de uma plataforma capaz de criar e gerenciar redes SDNs virtuais. Tanto a topologia quando o esquema de endereçamento de tais redes virtuais são completamente independentes entre si. Este trabalho pode ser aplicado em ambientes que utilizam OVX, uma vez que a topologia física necessária para comportar a topologia virtual poderia se beneficiar da agregação de enlaces.

No artigo de [Vencioneck et al., 2014b], os autores apresentam uma nova plataforma para controlar o encaminhamento de pacotes substituindo o *Open vSwitch*. Apesar de ser inovador e possuir resultados promissores, esse estudo é complementar a este trabalho, pois não considera a agregação de enlaces entre dois dispositivos.

Os autores [Raiciu et al., 2011] propõem a implementação de uma versão melhorada do protocolo TCP em ambientes de *data centers*, a fim de realizar a transmissão de pacotes utilizando vários caminhos. É uma abordagem semelhante à este trabalho, mas utiliza uma técnica diferente à agregação de enlaces. Da mesma maneira, [Vencioneck et al., 2014a] apresentam uma plataforma chamada *FlexForward* que também é direcionada para ambientes de *data centers*. Porém, sua implementação é focada numa alteração do *Open vSwitch*.

Capítulo 4

Metodologia

Esta seção apresenta a arquitetura, implementação, técnica de identificação dos enlaces de agregação e os algoritmos de agregação de enlace proposta neste trabalho.

4.1 Arquitetura

A figura 4.1 retrata a arquitetura proposta:

- **Aplicação (01)**: Nesta camada, foram criadas as regras de agregação de enlace. Exemplo: as políticas de encaminhamento de pacotes de acordo com a utilização de cada interface. No total foram criadas três aplicações, que são representadas por três programas na linguagem *Python* que são definidas nos parâmetros de entrada quando o controlador é iniciado.

- **Interface de ligação norte (*Northbound*) (02)**: API de comunicação entre o controlador e as aplicações de rede. Usualmente controlada e gerenciada pelo próprio controlador.

- **Controlador (03)**: Responsável por gerenciar a rede local. Executa as decisões da aplicação sobre as políticas de roteamento, encaminhamento, redirecionamento e balanceamento de carga. O controlador envia regras para a tabela de fluxos dos *switches*. É utilizado o controlador *Ryu*.

- **Interface de ligação sul (*Southbound*) (04)**: Protocolo de comunicação entre o controlador e os equipamentos de rede. Utiliza o padrão *OpenFlow*, já que existem funções pré-definidas no controlador *Ryu* que facilitaram o desenvolvimento das aplicações.

- ***Switch* (05)**: responsável por comutar os pacotes. Em cada equipamento de rede, o cliente *OpenFlow* é configurado utilizando a tabela de encaminhamento de pacotes. No ambiente desta pesquisa, são criadas instâncias virtuais do *Open vSwitch* pela plataforma *Mininet*.

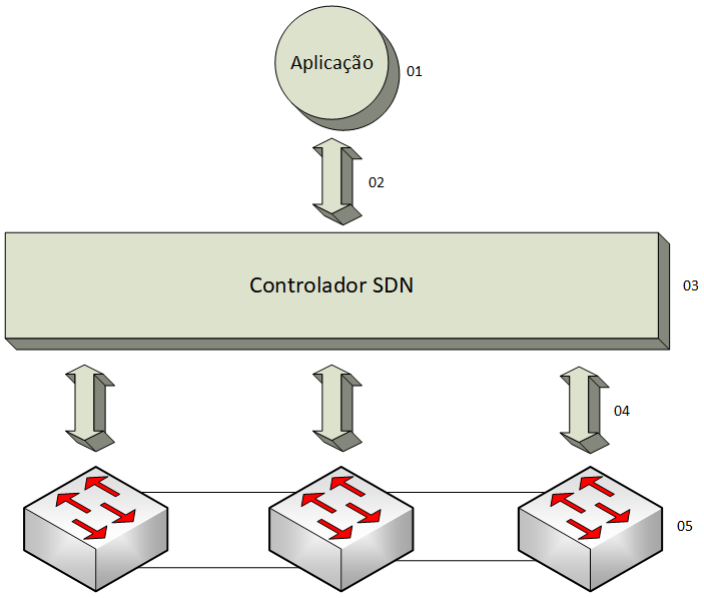

Figura 4.1. Arquitetura proposta neste trabalho

4.2 Implementação

O sistema foi desenvolvido utilizando o controlador *Ryu*, que gerencia todos os detalhes das conexões entre os *switches*, interpretando e convertendo os pacotes de rede em estruturas de dados fáceis de usar. Aplicações desenvolvidas para rodar utilizando este controlador utilizam apenas um processo no sistema operacional e as mensagens trocadas entre o controlador e os equipamentos de redes são assíncronas. Cada aplicação possui uma fila para tratar os eventos recebidos, sendo que esses eventos são computados de forma *FIFO* (*First In, First Out*). A cada item da fila tratado, o processo invoca o *handler* apropriado (*PacketOut* para enviar pacotes, *LinkDown* quando uma interface é desabilitada, etc). Enquanto cada evento é processado, outros eventos recebidos pelo controlador ficam em espera até que aquela posição da fila seja liberada.

O aplicativo desenvolvido escuta as mensagens de pacotes ingressantes no *switch*, mantém uma tabela interna de endereços MAC de todas as estações conectadas e adiciona as regras de encaminhamento nos *switches* à medida em que as conexões são identificadas. Para isso, o protocolo *OpenFlow* é utilizado.

Basicamente, a aplicação executa os seguintes passos:

1. **Registro e inicialização da aplicação**: Acontece antes dos *switches* serem inseridos no domínio do controlador *Ryu* e permite que a instância da aplicação (ou processo do sistema operacional) inicialize os dados que serão compartilhados em toda a rede. Por exemplo, uma tabela para manter informações dos endereços MAC das estações que estarão conectadas nos *switches* é inicializada sem dados, mapeando as respectivas portas às quais elas estarão conectadas.

2. **Inicialização de um *switch* que se conecta ao controlador *Ryu***: Quando um *switch* é inserido no domínio do controlador *Ryu*, o aplicativo recebe um evento que verifica todas as suas características. Assim que ele é recebido, uma regra de encaminhamento estática é adicionada ao *switch*. Essa regra faz com que qualquer pacote seja encaminhado para o controlador, enquanto não houver regras na tabela de fluxos para direcioná-lo.

3. **Identificação das agregações de enlaces**: A implementação presente neste trabalho não depende de nenhum protocolo para identificar as agregações entre um par de *switches*. Vários fabricantes utilizam o LLDP para realizar tal função.

4. **Envio de pacotes**: O aplicativo monitora o envio de pacotes quando o *switch* não possui informações suficientes em suas tabelas internas para realizar a comutação.

A atual implementação utiliza como base o código da aplicação *simple switch* existente no pacote padrão do controlador *Ryu*. Dessa forma, toda a inteligência e tomada de decisão foram implementadas diretamente no controlador, fazendo com que o *switch* seja apenas responsável pela comutação de pacotes de acordo com as regras existentes em sua memória em tempo de execução. Tal aplicação possui duas partes principais: a primeira responsável pelo processo automático de identificação de enlaces agregados, e a segunda responsável pela implementação das políticas de comutação de pacotes.

4.2.1 Principais eventos processados pelo controlador *Ryu*

A seguir, os principais eventos utilizados pelo controlador para o desenvolvimento deste trabalho são detalhados. Em resumo, todos os eventos gerados tanto *switch* quanto pelo controlador são assíncronos. Dessa forma, nenhuma das partes envolvidas precisa aguardar pelo retorno das informações solicitadas. Vale ressaltar também que a versão 1.4 do *OpenFlow* foi utilizada.

1. **OFPFeaturesRequest**: Quando um *switch* estabelece uma conexão no domínio do controlador, este evento é enviado para ele. O seu objetivo é coletar informações básicas, tais como o ID e número de tabelas existentes.

2. **EventOFPSwitchFeatures**: É o evento resposta gerado pelo *switch* ao receber o evento *OFPFeaturesRequest*.

3. **EventDP**: Evento enviado do *switch* para o controlador contendo informações sobre as portas locais (índices e endereços físicos).

4. **EventLinkAdd**: Evento gerado quando um novo enlace é adicionado em um *switch*. Este enlace é necessariamente a conexão entre dois *switches*, já que a conexão entre estações e *switches* é tratada por outro evento. É através deste evento que os grupos das agregações de enlaces são criados.

5. **EventLinkDelete**: Evento oposto ao *EventLinkAdd*, ocorre quando um enlace entre *switches* é removido. É utilizado para remover enlaces das agregações.

6. **EventOFPPacketIn**: Quando o *switch* não possui informações suficientes em suas tabelas internas para realizar a comutação de pacotes, ele envia esse evento com as informações do pacote para o controlador. É através deste evento que as políticas de agregação de enlaces são configuradas.

7. **EventOFPErrorMsg**: Evento gerado pelo controlador quando existe algum parâmetro incorreto nas mensagens trocadas entre ele e o *switch* (por exemplo, a criação de um fluxo foi enviada para o *switch* sem a informação da porta de destino). Utilizado para *debugar* o sistema como um todo.

4.3 Identificação das agregações de enlaces

Quando o controlador SDN recebe a notificação da inclusão de um *switch* em seu domínio, ele verifica os enlaces desse *switch* com seus *switches* vizinhos. Caso sejam identificadas mais de uma ligação entre um mesmo par de *switches*, o controlador assume que existe uma agregação de enlaces e cria uma estrutura de dados para armazenar essa informação. Tal estrutura é criada utilizando uma estrutura de dados, onde o índice representa os IDs dos *switches* e os valores representam interfaces de cada *switch* que fazem parte da agregação. Essa estrutura é representada pela figura 4.2. Com esse mecanismo, não há a necessidade da configuração manual por parte de um administrador de redes.

```
{
 (6, 7):
        {
         6: [5, 7, 6, 8],
         7: [5, 7, 6, 8]
        },
 (4, 6): {
          4: [6, 5],
          6: [4, 3]
         },
 (5, 7): {
          5: [5, 6, 8, 7],
          7: [1, 2, 4, 3]
         },
 (1, 5): {
          1: [6, 5],
          5: [2, 1]
         },
 (3, 6): {
          3: [6, 5],
          6: [2, 1]
         },
 (2, 5): {
          2: [5, 6],
          5: [3, 4]
          }
}
```

Figura 4.2. Exemplo da estrutura de dados das agregações identificadas

4.4 Mecanismo de prevenção de *loop*

Em ambientes de redes, quando um pacote é retransmitido indefinidamente, um efeito em cascata é gerado. Sempre que um *switch* não sabe a porta de destino para enviar um determinado pacote, ele é enviado para todas as suas interfaces com exceção da interface de origem. Esse é um mecanismo comum em ambientes de redes e utilizado por protocolos como o ARP). Quando existe mais de uma interface ligada entre dois *switches* e ambas estão ativas (sem a configuração do *Spanning Tree*, por exemplo), em um determinado momento o pacote será enviado de volta para o *switch*. Por não existir um campo de tempo de vida na camada 2 do protocolo de redes, várias cópias deste pacote serão encaminhadas e reencaminhadas entre os *switches*, eventualmente causando um estouro de memória e exaustão da CPU dos equipamentos e paralisando completamente a rede de computadores. Conhecido como *broadcast storm* e representado pela figura 4.3, este grave problema é prevenido utilizando uma ideia simples: mesmo com todos os pacotes dos tipos *broadcast* e *multicast* sendo encaminhados para todas as portas do *switch* que recebeu o tráfego de origem, caso o *switch* destino receba estes pacotes em portas de uma agregação, apenas aqueles oriundos da primeira porta do grupo serão processados. Por exemplo, caso um determinado *switch* possua uma agregação com as portas 1, 2 e 3, todos os pacotes *broadcast* e *multicast* recebidos pelas portas 2 e 3 serão ignorados. A figura 4.4 exibe o trecho do código da implementação que previne este problema.

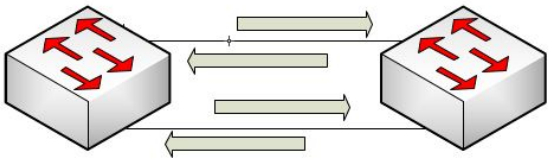

Figura 4.3. Exemplo básico de um *broadcast storm*

```
# Loop prevention mechanism: only accepts broadcast and multicast packets received by the firt interface of the aggregation
if (in_port in broadcast_ports[1:]) and ("01:00:5E:" in mac_dst or "33:33:" in mac_dst or mac_dst == "ff:ff:ff:ff:ff:ff"):
    return
```

Figura 4.4. Código do mecanismo de prevenção de *loop*

4.5 Método de prevenção de falhas

No caso de uma interface da agregação ficar indisponível por qualquer motivo (um cabo desconectado ou interface desabilitada, por exemplo), o algoritmo é capaz de identificar essa falha e adaptar o grupo removendo a interface problemática. Quando o oposto acontece (uma interface nova é adicionada na agregação ou interface com problemas volta a funcionar), o grupo da agregação é atualizado automaticamente com a nova interface. Quando uma interface volta a ficar disponível, ela é tratada como uma nova interface da agregação. Isso faz com que novos fluxos possam ser criados ou recriados utilizando ela, dependendo do algoritmo de agregação utilizado.

4.6 Tráfego intra-fluxo

Um tráfego intra-fluxo consiste em enviar os pacotes do mesmo fluxo entre interfaces diferentes. Existem algumas formas de executar tal ação, como utilizar o número de sequência em pacotes TCP ou criar um contador interno para definir a próxima interface de destino, mas realizar tal tarefa através do protocolo *Openflow* não é possível. Até o presente momento, nenhum dos campos existentes na versão mais atual do *Openflow* (1.5.1) é capaz de realizar a separação de pacotes intra-fluxo. O *switch Open vSwitch* também não provê mecanismo para isso. Por isso, foi utilizado o projeto BPFabric [Jouet & Pezaros, 2017], que permite instalar instruções eBPF para serem executadas e armazenar estado através de tabelas dentro de um *switch* programável.

Capítulo 5

Algoritmos de agregação de enlace

Em redes SDN, existe a liberdade de implementar um algoritmo customizado de agregação de enlaces. É possível classificar os algoritmos de agregação como do tipo inter-fluxo ou intra-fluxo. O tipo inter-fluxo ocorre quando cada fluxo é separado em caminhos distintos, mas não há divisão do fluxo. O tipo intra-fluxo ocorre quando existe a divisão de um fluxo em vários sub-fluxos com caminhos distintos. Diante disto, durante este trabalho, as agregações foram identificadas e criadas utilizando três técnicas distintas que serão descritas a seguir.

O primeiro algoritmo implementado, chamado Tabela *Hash*, funciona de forma parecida com o LACP. Durante sua execução, todos os fluxos entre o mesmo par de estações utilizarão a mesma interface. O segundo, chamado Análise de Tráfego, prioriza a utilização de interfaces com baixo tráfego para a tomada de decisão de qual interface utilizar. Por fim, o algoritmo *Round-Robin* Virtual utiliza todas as interfaces da agregação modificando as prioridades dos fluxos periodicamente.

Os dois primeiros algoritmos são considerados inter-fluxos, e o último pode ser considerado um híbrido de inter-fluxo e intra-fluxo. Isso se deve ao fato de que a transmissão de pacotes passa em blocos por cada interface da agregação a cada intervalo de tempo e para ser considerado intra-fluxo, cada pacote deveria passar por interfaces diferentes. Por fim, vale ressaltar que a plataforma de emulação *Mininet* não suporta a configuração do protocolo LACP entre dois *switches* virtuais. Por esse motivo, foi implementado de um algoritmo parecido com o LACP, batizado de Tabela *Hash*, para superar essa limitação.

5.1 Tabela *Hash*

As estações ligadas nos *switches* começam a transmitir dados. Quando o *switch* recebe o primeiro pacote do fluxo, ele o retransmite para o controlador, que identifica os campos disponíveis (endereço MAC de origem e destino, endereço IP de origem e destino) e calcula um *hash*, que determina qual interface será utilizada para transmitir os dados daquele fluxo. Essa escolha poderia ser aleatória, mas acredita-se que a distribuição não seria tão eficaz com quantidades pequenas de interfaces agregadas. O controlador insere a regra de encaminhamento no *switch*, que encaminha os pacotes daquele fluxo pela interface calculada pelo controlador. O algoritmo 1 representa a implementação desse procedimento.

Algorithm 1 Algoritmo implementando a política Tabela **Hash**

```
1: function CALCULA HASH(pacote)
2:     valor = hash(mac origem, mac destino, ip origem, ip destino)
3:     return valor
4: end function
```

5.2 Análise de Tráfego

O *switch*, ao receber o primeiro pacote do fluxo, o retransmite para o controlador, que identifica a utilização da banda de cada interface da agregação para tomar a decisão de qual interface será utilizada para transmitir os dados do fluxo em questão. A prioridade é escolher interfaces com baixa utilização de banda, permitindo assim uma melhor distribuição do tráfego entre todas as interfaces. Apesar desta escolha ser realizada apenas no começo da transmissão do tráfego, eventualmente ela será refeita já que as regras criadas possuem um tempo de expiração de 30 segundos após ocorrer inatividade. O controlador insere a regra de encaminhamento no *switch*. Em seguida, o *switch* encaminha os pacotes daquele fluxo pela interface escolhida pelo controlador. Tal técnica é classificada como do tipo inter-fluxo. O algoritmo 2 representa a implementação desse procedimento.

Algorithm 2 Algoritmo implementando a política **Análise de Tráfego**

```
1: function INTERFACE COM MENOR TRAFEGO(switch)
2:     Recupera o tráfego de todas as interfaces
3:     Identifica a interface com menor utilização de tráfego
4:     return interface
5: end function
```

5.3 *Round-Robin* virtual

A diferença entre esta técnica e as duas anteriores é a quantidade de regras criadas. Ao invés de criar regras para apenas um fluxo para cada transmissão de uma origem para um destino, o controlador cria regras para todas as origens e todos os destinos utilizando todas as interfaces da agregação. Porém, cada regra é criada com prioridades diferentes, fazendo com que o *switch* escolha apenas uma interface para transmitir os pacotes. Por exemplo, em uma topologia com dois *switches*, uma estação em cada *switch* e duas interfaces ligadas entres eles, um total de oito fluxos seriam criados. Dois fluxos entre cada par de estações, o primeiro com uma prioridade 1 e o segundo com prioridade 50, sendo que o algoritmo altera essas prioridades de tempos em tempos, modificando o caminho por onde os pacotes passam.

O intervalo de atualização foi fixado em 0,2 segundos e foi obtido experimentalmente com base no melhor Índice de Justiça [Jain et al., 1984] resultando, que é um indicador utilizado para determinar se os usuários ou aplicativos estão recebendo uma parcela justa de banda da rede. Essa técnica pode ser classificada como híbrida, já que possui traços de inter-fluxo e intra-fluxo. Como o tráfego muda de caminho a cada 0,2 segundos, cada parte do tráfego total passará por uma interface diferente. Sendo assim, essa técnica não é totalmente inter-fluxo (o trafégo total não passa apenas por uma interface da agregação) e nem intra-fluxo (cada pacote não é enviado para interfaces diferentes, mas sim conjuntos de pacotes). O algoritmo 3 representa a implementação desse procedimento.

Vale ressaltar que a quantidade de regras criadas possui um aumento exponencial, já que depende da quantidade de estações na topologia e quantidade de enlaces agregados. Porém tanto no ambiente virtual quando no real propostos, a escalabilidade foi satisfatória.

Algorithm 3 Algoritmo implementando a política ***Round-Robin* virtual**

```
1: while True do
2:     próximo_índice ←(índice+1)%Número_de_Regras
3:     Regra[próximo_índice].prioridade ← 50;     ▷ Aumenta prioridade da próxima regra
4:     Regra[índice].prioridade ←1;               ▷ Diminui prioridade da regra atual
5:     índice ← próximo_índice;                   ▷ Atualiza índice da regra atual
6:     Sleep τ ms
7: end while
```

Capítulo 6

Avaliação Experimental e Resultados

O objetivo desta pesquisa inclui abordagens práticas e experimentais. Para atingir o primeiro objetivo, um ambiente virtual com os seguintes componentes foi criado:

- **Mininet**: plataforma capaz de criar redes de computadores virtuais e complexas, auxiliando nas simulações das transmissões de dados. A versão 2.3.0d1 mais recente foi utilizada.

- **OpenFlow**: protocolo de comunicação utilizado entre equipamentos SDN. A versão 1.4 está disponível no controlador *Ryu* e foi utilizada.

- ***Ryu* Framework**: controlador SDN que utiliza a linguagem de programação Python, atualmente na versão 4.18.

O experimento virtual foi dividido em dois tipos. No primeiro, a ferramenta *iperf* foi utilizada para gerar tráfego TCP e UDP entre 16 estações na topologia. O *iperf* não considera a sobrecarga de transferência de cabeçalhos de protocolo como *Ethernet*, IP, UDP ou TCP. Portanto, nos resultados apresentados, a largura de banda máxima teórica disponível não foi alcançada. O segundo conjunto de testes foi realizado usando a ferramenta *tcpreplay* usando dois arquivos de tráfego realistas: um capturado durante uma transferência de arquivos e outro capturado durante uma atividade de navegação na internet.

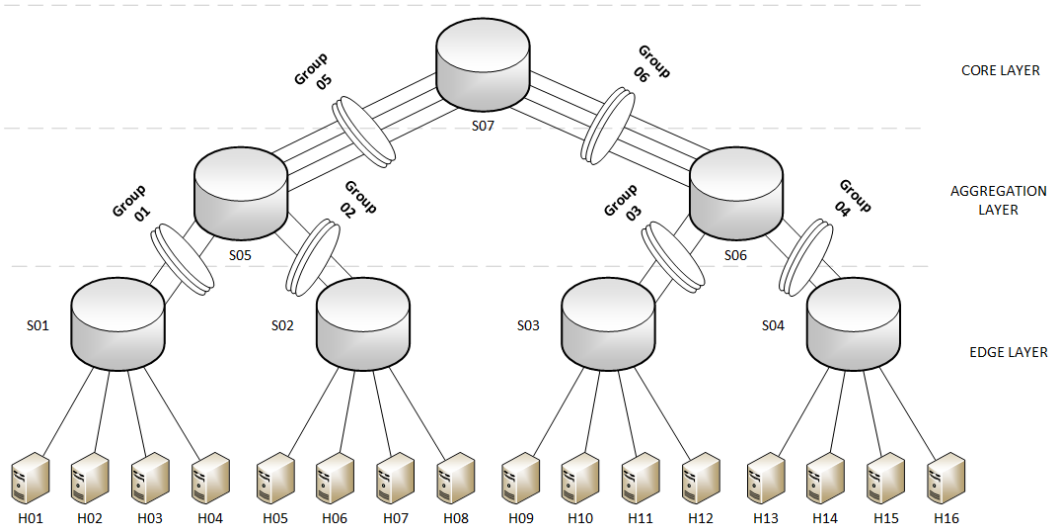

Figura 6.1. Topologia virtual com agregação de enlaces

6.1 Topologia virtual

Para avaliar e validar a esta pesquisa, foi criada uma topologia inspirada na topologia de *Fat-Tree* clássica [Leiserson, 1985]. Com um total de 7 *switches* e 16 hosts, a topologia consiste em três níveis: núcleo, borda e camadas de agregação. Entre cada camada, dois ou mais enlaces foram conectados entre os *switches* para fornecer melhor rendimento para a transmissão de dados, enquanto todos foram configurados com uma velocidade de 100 Mbps. A figura 6.1 mostra esta topologia.

6.1.1 Ambiente virtual

O ambiente virtual foi construído usando uma máquina virtual na plataforma da nuvem do Google. Uma vez que apenas uma configuração básica era necessária, uma VM com dois processadores Intel 2.30 GHz, 8 GB de RAM e 10 Gb de espaço em disco foi criada.

6.1.2 Banda total utilizada

Com o objetivo de calcular a transmissão máxima de dados na topologia virtual, a ferramenta *iperf* foi escolhida para criar fluxos as oito primeiras estações para as últimas oito. As transmissões TCP e UDP foram feitas, e os três algoritmos foram testados. Neste cenário, vale a pena mencionar que o rendimento máximo seria de 400 Mbps. A tabela 6.1 representa os resultados desses testes.

Através dos resultados obtidos nos testes do tráfego gerado artificialmente, percebe-se uma diferença entre vazão total e distribuição justa do tráfego dos pa-

	01	02	03	04	05	06	07	08	Total	Índice de justiça
Tabela Hash (TCP)	52.1	59.9	23.4	53.7	18.2	54.5	56.0	66.4	384.2	89.66%
Tabela Hash (UDP)	86.5	87.1	2.9	39.8	10.4	91.6	43.6	5.8	367.7	62.38%
Análise de tráfego (TCP)	36.4	41.3	33.5	45.8	23.7	24.6	39.6	44.8	289.7	95.43%
Análise de tráfegos(UDP)	10.6	15.7	81.5	96.7	63.3	13.3	4.8	11.8	297.7	53.52%
Round-Robin virtual(TCP)	47.8	47.8	47.8	47.8	37.9	47.9	47.8	10.6	335.4	92.10%
Round-Robin virtual(UDP)	24.7	48.6	22.4	48.6	48.6	24.0	48.6	26.3	291.8	89.99%

Tabela 6.1. Tráfego gerado artificialmente: banda total medida por cada estação transmissora (em Mbps)

cotes. Considerando uma vazão maior, os testes utilizando o algoritmo Tabela *Hash* obtiveram resultados superiores. Porém, ao considerar a distribuição justa do tráfego, o algoritmo Análise de Tráfego foi o melhor para tráfego TCP e *Round-Robin* virtual para tráfego UDP. A escolha do melhor algoritmo nesse caso depende da métrica a ser considerada (vazão ou distribuição), porém ao observar as duas em paralelo, o algoritmo *Round-Robin* virtual obteve o melhor resultado.

A figura 6.2 ilustra a vazão total entre os tráfegos TCP e UDP dos três algoritmos implementados.

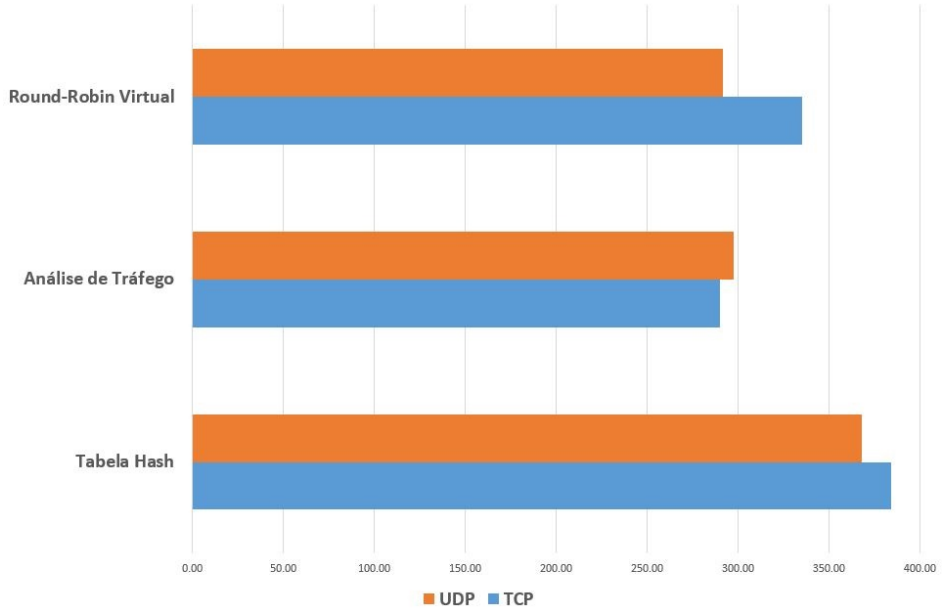

Figura 6.2. Comparativo entre tráfegos TCPxUDP entre os três algoritmos implementados

Uma vez que o tráfego UDP não precisa de confirmação (como o tráfego TCP), é de se esperar que ele tenha uma maior vazão. Isso ocorreu com dois dos três algoritmos e o motivo de tal comportamento, assim como a vazão abaixo do máximo disponível,

é explicado a seguir.

Vale ressaltar que, por motivos técnicos[1] não é possível alcançar a vazão máxima em enlaces *ethernet*. Em um enlace de 1 Gbps, por exemplo, a vazão máxima é de apenas 761.90 Mbps. Além disso, por se tratar de um ambiente virtual, a implementação e funcionamento das máquinas virtuais podem impactar nos testes.

6.1.3 Tráfego realista

Como o *iperf* não pode simular cenários de tráfego realistas, outra ferramenta foi utilizada para atingir esse objetivo. A ferramenta *tcpreplay* é capaz de reconstruir e retransmitir um determinado arquivo *pcap* (arquivo geralmente criada durante a captura de pacotes de redes) de uma origem para um destino. Com essa habilidade, um conjunto de testes simulando uma navegação na Internet foi realizado, gerando fluxos das primeiras oito estações para as últimas oito. A tabela 6.2 representa os resultados desses testes. Apenas as interfaces do *switch* 07 foram observadas durante este teste, já que este é o *switch* central da topologia e todo o tráfego passa por ele. Considerando que cada lado deve representar uma quantidade total de 100 % (lado esquerdo com as interfaces 01, 02, 03 e 04, enquanto o lado direito possui as interfaces 05, 06, 07 e 08), os valores representam a vazão total de cada interface individual.

	01	**02**	**03**	**04**	**Total**	**Índice de justiça**	**05**	**06**	**07**	**08**	**Total**	**Índice de justiça**
Tabela Hash	58.2	43.6	28.8	47.9	178.5	94.69%	48.1	46.7	20.8	45.7	161.3	92.71%
Análise de tráfego	54.4	39.1	18.2	37.1	148.8	89.34%	46.6	26.7	8.9	25.2	107.4	80.13%
Round-Robin virtual	39.8	38.2	53.7	26.5	158.2	94.38%	22.1	32.4	39.5	32.4	126.4	96.29%

Tabela 6.2. Tráfego realista: banda total medido por cada porta (em Mbps)

6.1.4 Queda das interfaces

Para verificar o comportamento da plataforma durante a queda de uma interface qualquer, o seguinte teste foi realizado:

- Utilizando a ferramenta *iperf*, foram criados fluxos de dados entre as quatro primeiras e as quatro últimas estações
- Observando a carga das interfaces de entrada que conectam o *switch* 07 com o *switch* 05, após 15 segundos de transmissão, a interface 01 da agregação foi desativada.

[1] `https://goo.gl/FGzK7z`

- Após 6 segundos, a mesma interface foi reativada e o tráfego foi encerrado após 30 segundos de transmissão.

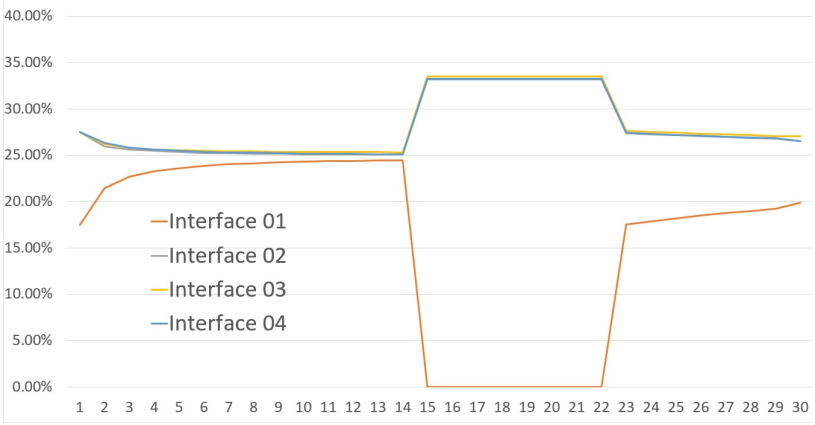

Figura 6.3. Utilização das interfaces durante a queda da interface 01

A figura 6.3 exibe a carga de tráfego distribuída entre as quatro interfaces da agregação. É notável que durante a queda da interface 01, a carga é igualmente distribuída entre outras interfaces da agregação. Quando essa interface volta a ficar disponível, o tráfego volta a ficar distribuído igualitariamente entre todas as interfaces da agregação. Além disso, percebe-se que houve uma certa diferença entre a distribuição do tráfego no início da transmissão e após a interface voltar a ficar disponível. Por se tratar de um ambiente virtual, onde os recursos da máquina real são compartilhados entre várias máquinas virtuais, comportamentos estranhos como este podem acontecer.

6.1.5 Tráfego intra-fluxo

A versão do *OpenFlow* utilizada neste trabalho possui uma variedade de campos para a correspondência de fluxo. Dentre 42 opções, apenas 5 foram utilizadas para implementar os algoritmos anteriores, chamados inter-fluxo. Até o presente momento, nenhum dos campos existentes na versão utilizada neste trabalho ou na versão mais atual do *OpenFlow* (1.5.1) é capaz de realizar a separação de pacotes intra-fluxo.

Para tal, é necessário realizar edições diretamente no código do *switch*. Por se tratar de uma implementação extremamente complexa e extensa, o *Open vSwitch* não foi utilizado para provar o conceito de transmissões intra-fluxo utilizando interfaces agregadas. Foi desenvolvido, em linguagem C, uma aplicação que armazena no *switch* um contador de pacotes por fluxos. Para todo pacote recebido, é incrementado o contador de seu respectivo fluxo. O encaminhamento do pacote depende do valor do contador de pacotes. O algoritmo utilizado foi o do *round-robin*. É calculado o resto da

divisão (mod) do contador de pacotes pelo número de enlaces na agregação. O switch já possui a instrução mod no seu plano de dados.

O programa em C é compilado para a plataforma eBPF pelo compilador LLVM 3.9. Depois o conjunto de instruções é extraído pela ferramenta *objdump* do executável que tinha sido gerado pelo compilador. Finalmente, o conjunto de instrução é instalado dentro do *switch* que executa o código pela máquina virtual eBPF. Desta forma, foi possível projetar e construir um *switch* que possui capacidade de realizar agregação intra-fluxo.

A topologia simples da figura 6.4 foi implementada no *Mininet*, e os dados da estação transmissora e receptora foram incluídos diretamente no código. Isso faz com que esse teste funcione apenas neste cenário, porém através deste teste é o possível provar o funcionamento da transmissão do tráfego intra-fluxo.

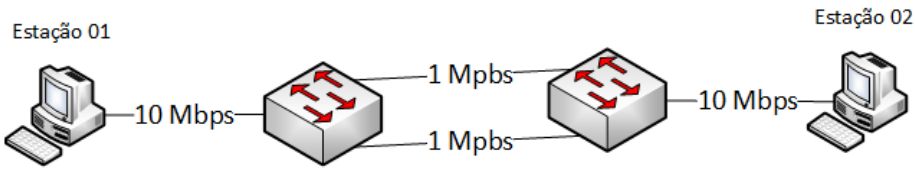

Figura 6.4. Topologia utilizada para provar a teoria do intra-fluxo

O seguinte teste foi realizado:

- Um enlace de 10 Mbps foi criado entre as estações e os *switches*.
- Entre os dois *switches*, dois enlaces de 1 Mbps cada foram criados.
- Utilizando a ferramenta *iperf*, a vazão do tráfego entre a estação 01 e 02 foi medido.

A escolha de configurar o enlace entre os *switches* com uma banda de apenas 1 Mbps, uma banda menor do que o enlace entre as estações e os *switches*, é proposital. Dessa forma, o funcionamento da política intra-fluxo é observada caso a vazão seja próxima dos dois enlaces existentes.

O primeiro teste realizado utilizou apenas uma interface conectada entre os dois *switches*. Conforme o esperado, a banda total utilizada foi de 958 Kbps ficando um pouco abaixo da banda disponível.

Já o segundo teste utilizou as duas interfaces agregadas da topologia. Desta vez, a banda total utilizada foi de 1,91 Mbps. Isso prova que a banda das duas interfaces da agregação foi utilizada. Além deste resultado, utilizamos a ferramenta *tcpdump* para

verificar se os pacotes foram separados igualitariamente entre as duas interfaces. A figura 6.5 exibe um trecho dos pacotes coletados, mais uma vez provando que o intra-fluxo foi realizado com sucesso. Nela, percebe-se que a primeira interface transmitiu o pacote com número de sequência 13057 e a segunda o pacote com 14505.

Vale ressaltar que testes de latência também foram realizados, a fim de determinar o impacto da variabilidade de atraso da entrega dos pacotes enviados de uma estação para outra. O valor médio do RTT (*Round Trip Time*, ou tempo de ida e volta do pacote) foi de 0.040 milissegundos. Isso ocorre porque os pacotes passam pelos mesmos *switches*, sendo que a única diferença é a interface de rede.

6.2 Experimento com laboratório real

Como prova de conceito para demonstrar a aplicação desta pesquisa, os algoritmos foram executados em equipamentos reais. Para realizar este marco, o seguinte laboratório real foi configurado:

- **Switch Openflow**: roteadores sem fio de baixo custo com um firmware personalizado foram comprados para criar o laboratório. O hardware escolhido foi o modelo WR841N da fabricante *TP-Link*, que possui apenas 4 Mb de ROM, 32 Mb de RAM e 5 portas *Fast Ethernet* (uma para conectividade WAN e quatro para conectividade LAN). Devido às suas limitações de memória, um firmware personalizado foi construído usando o Projeto LEDE, para acomodar o *Open vSwitch*. A remoção de pacotes nativos do *switch* (como suporte sem fio, PPP,

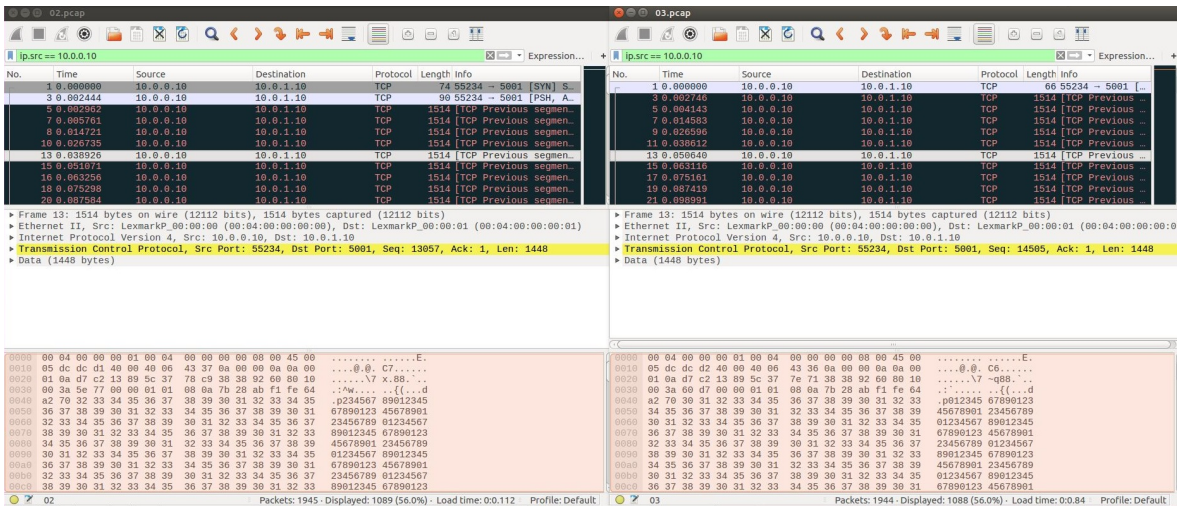

Figura 6.5. Imagem da coleta de tráfego realizada entre as interfaces agregadas

DHCP e DNS) foi necessária para ser possível instalar e executar o *Open vSwitch*. Tais remoções foram realizadas apenas para liberar espaço na memória ROM do roteador e não afetaram sua funcionalidade. Apenas dois desses roteadores foram usados, conectando as quatro interfaces LAN entre si, criando um único grupo de agregação.

- **Controlador**: para controlar cada *switch*, um *Rasperry Pi* foi conectado à sua porta WAN. O sistema operacional *Raspbian* foi instalado, assim como a versão mais recente do controlador *Ryu*.

- **Estações**: Como o *Rasperry Pi* possui apenas uma interface física, interfaces virtuais foram criadas e configuradas para simular quatro estações conectadas em cada switch.

6.2.1 LEDE

O projeto LEDE[2] (*Linux Embedded Development Environment*) é um sistema operacional *Linux* baseado no famoso *OpenWrt*[3]. Ambos são utilizados para criar *firmwares* customizados de diversos fabricantes de roteadores sem fio. Através destes *firmwares*, além de uma maior flexibilidade na configuração dos equipamentos, uma infinidade de novos pacotes podem ser instalados. Com isso, é possível instalar o *Open vSwitch* em pequenos roteadores residenciais, que chegam a custar em torno de R$80,00.

O processo de criação do *firmware* customizado do LEDE para um determinado modelo de roteador sem fio é bastante simples[4] e a figura 6.6 exibe a tela inicial do criador de imagens. O pré-requisito inicial consiste em possuir uma ambiente *Linux*, podendo até mesmo ser virtual.

6.2.2 Tráfego real

Idêntico aos testes de topologia virtual, o teste de tráfego realista realizado no laboratório da vida real também utiliza o *tcpreplay*, bem como os mesmos arquivos *pcap*. O teste consistiu em gerar fluxos dos primeiros quatro hosts para os últimos quatro, simulando uma navegação na Internet mais uma vez. A figura 6.7 ilustra a topologia do laboratório, enquanto que a figura 6.8 exibe os equipamentos utilizados.

Além dos algoritmos implementados e já citados anteriormente, outros dois algoritmos foram utilizados durantes os testes do laboratório com equipamentos reais:

[2] `https://lede-project.org/`
[3] `https://openwrt.org/`
[4] https://lede-project.org/docs/user-guide/imagebuilder

o *Linux Bonding* nas versões SLB e TCP. As diferenças entre ambos são descritas a seguir:

- **Linux Bonding SLB**: Um *hash* é criado utilizando os valores da VLAN e dos endereços MAC de origem e destino, fazendo com que todos os pacotes de cada par de estações sejam enviados no mesmo enlace. Esta configuração não necessita da ativação do protocolo LACP.

- **Linux Bonding TCP**: Todos os campos das camadas 2, 3 e 4 são utilizados para criar um *hash* nesta configuração (endereços MAC, IP e portas de origem e destino). Esta configuração necessita da ativação do protocolo LACP, que nesse cenário é controlado e ativado pelo *Open vSwitch*.

O *Linux Bonding* é uma biblioteca nativa existente em equipamentos que utilizam o sistema operacional *Linux* e geralmente é utilizado nas conexões entre servidores/servidores e servidores/switches. Ao utilizar essa biblioteca, o usuário precisa especificar qual premissa será utilizada. As opções variam entre *balance-rr*, *active-backup*, *balance-xor*, *broadcast* e as já citadas *balance-slb* e *balance-tcp*. Vale ressaltar que nem todas essas opções estão disponíveis no *Open vSwitch*.

A tabela 6.3 representa os resultados do teste.

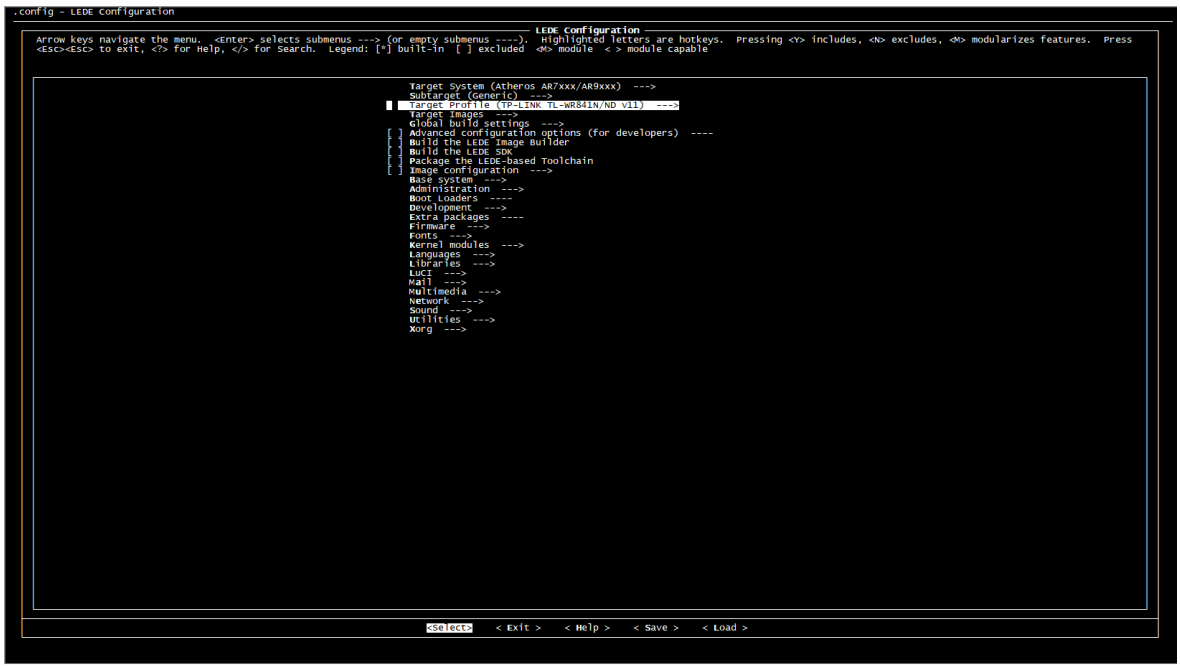

Figura 6.6. Tela principal do projeto LEDE

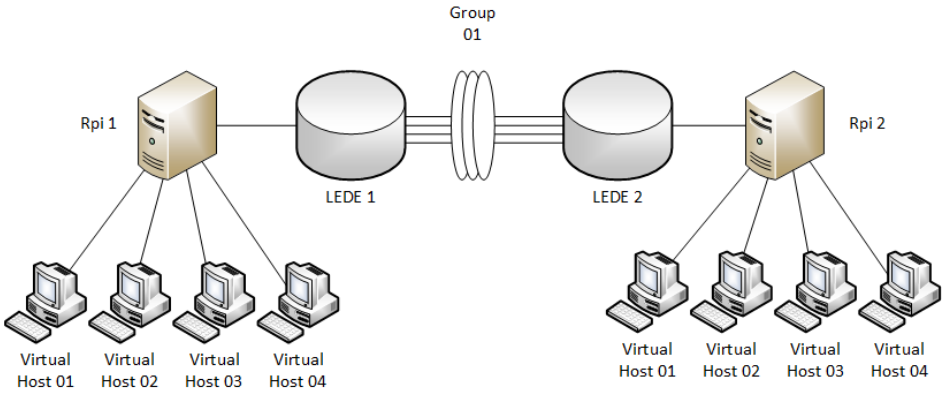

Figura 6.7. Laboratório real com agregação de enlaces

	01	**02**	**03**	**04**	**Total**	**Índice de justiça**
Linux Bonding (SLB)	22.7	22.6	22.9	22.6	90.8	100%
Linux Bonding (TCP)	10.9	55.6	9.6	12.3	88.4	56.57%
Tabela Hash	21.9	21.1	21.3	20.3	84.6	99.93%
Análise de tráfego	5.4	18.9	37.9	13.9	76.1	71.82%
Round-Robin virtual	21.8	21.9	21.8	21.9	87.4	100.00%

Tabela 6.3. Vazão total medida para cada transmissão de cada estação(em Mbps)

Através dos resultados obtidos, pode-se observar que tanto o algoritmo Tabela *Hash*, quanto o algoritmo *Round-Robin* virtual obtiveram resultados semelhantes ao algoritmo *Linux Bonding* SLB. Isso reafirma a necessidade de expandir os estudos de agregações de enlaces em redes SDN, já que o benefício de tal técnica pôde ser obser-

Figura 6.8. Equipamentos utilizados no laboratório real

vado. Ademais, a má distribuição da vazão no resultado do *Linux Bonding* TCP pode ser um reflexo da maneira como o laboratório foi montado (apenas um equipamento físico com várias interfaces virtuais em ambas as pontas), e testes adicionais precisam ser realizados com outras configurações para discutir melhor seu desempenho. Por fim, o ruim desempenho do algoritmo Análise de Tráfego evidencia a fragilidade da solução e a necessidade de melhorar o algoritmo ou a forma como ele funciona.

6.3 Discussão

As tabelas 6.1 e 6.2 também exibem o Índice de Justiça de [Jain et al., 1984], um indicador da distribuição da banda entre os enlaces agregados. Para facilitar o entendimento dos resultados, uma série de gráficos foram elaborados. A figura 6.9 representa a função de distribuição cumulativa (CDF) para o tráfego gerado artificialmente durante os testes no laboratório virtual. Nesta figura, nota-se que os melhores resultados foram obtidos pelos algoritmos Tabela *Hash* e *Round-Robin* virtual, ambos durante a transmissão de dados TCP. Considerando que a transmissão máxima de dados era de 400 Mbps, a maioria das estações nos testes deste dois algoritmos obtiveram uma taxa de transmissão entre 40 Mbps e 60 Mbps, o que representa um ótimo balanceamento de carga (considerando que, dividindo 400 Mbps entre oito enlaces, uma divisão perfeita seria cada enlace transmitindo 50 Mbps).

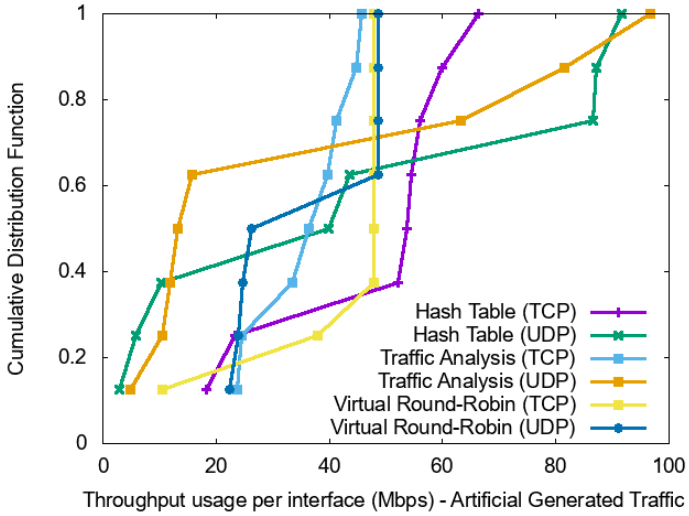

Figura 6.9. Função de Distribuição Cumulativa para os resultados de laboratório virtual (Tráfego artificial)

Na figura 6.10, a função CDF para o tráfego realista durantes os testes virtuais é representada. Neste contexto, os algoritmos Tabela *Hash* e *Round-Robin* virtual

obtiveram resultados bons. Por outro lado, o algoritmo Análise de Tráfego obteve um resultado abaixo do ideal, alcançando taxas de transmissão de dados entre 20 Mbps e 40 Mbps, onde o valor máximo era de 200 Mbps (considerando apenas as interfaces do lado esquerdo do *switch* 07).

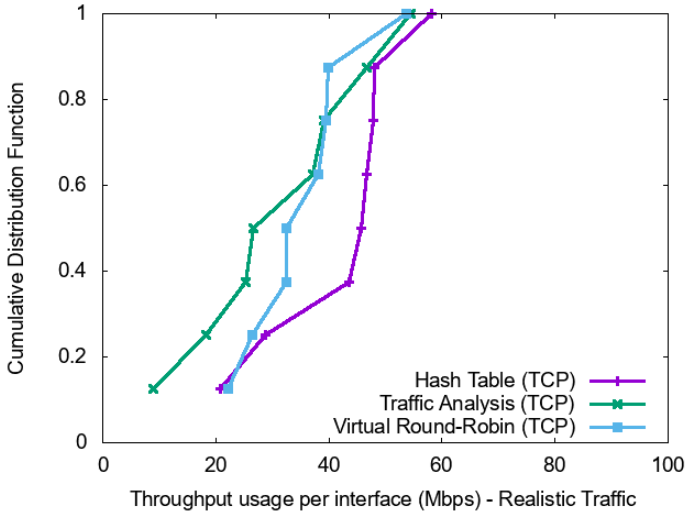

Figura 6.10. Função de Distribuição Cumulativa para os resultados de laboratório virtual (Tráfego real)

A figura 6.11 representa a função CDF para o tráfego realista durante os testes do laboratório real. Neste contexto, a transmissão máxima de dados era de 100 Mbps e nota-se que três algoritmos obtiveram resultados perfeitos: a versão SLB do *Linux Bounding*, o algoritmo Tabela *Hash* e o algoritmo *Round-Robin* virtual. Todos eles balancearam o tráfego de maneira uniforme entre todas as estações, consumindo em torno de 25 Mbps cada.

Por fim, a figura 6.12 representa a dispersão dos índices de justiça durante o teste do laboratório com o fluxo artificialmente gerado. Considerando a relação entre vazão máximo e melhor distribuição de tráfego, o algoritmo Tabela *Hash* obteve o melhor resultado durante a transmissão de dados TCP. Em seguida, pode-se observar o algoritmo *Round-Robin* virtual. Além disso, fica evidente o baixo desempenho obtido pelo algoritmo Análise de Tráfego nos dois tipos de dados.

O baixo desempenho do algoritmo de Análise de Tráfego em quase todos os testes demonstra que o código implementado precisa ser melhorado ou a política adotada não é adequada para utilização em um ambiente de agregação de enlaces. A forma como o enlace é escolhido no momento da criação da regra do fluxo também precisa ser avaliada. Provavelmente os pacotes iniciais são enviados para o controlador sem que o aumento do tráfego dos enlaces aumente consideravelmente, fazendo com que a mesma

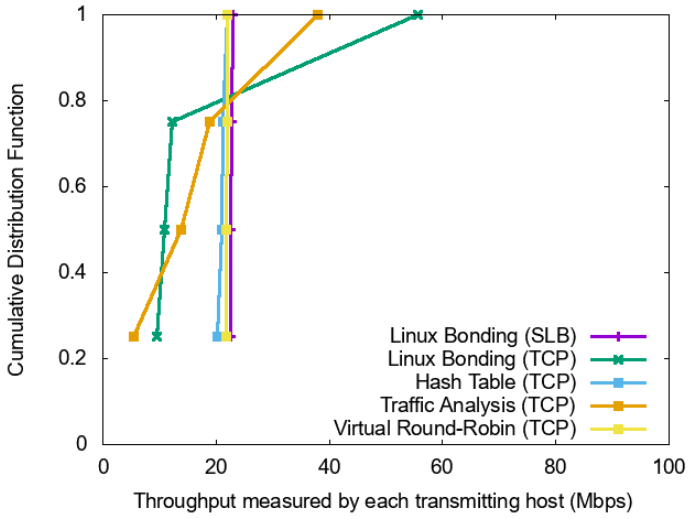

Figura 6.11. Função de Distribuição Cumulativa para os resultados da vida real

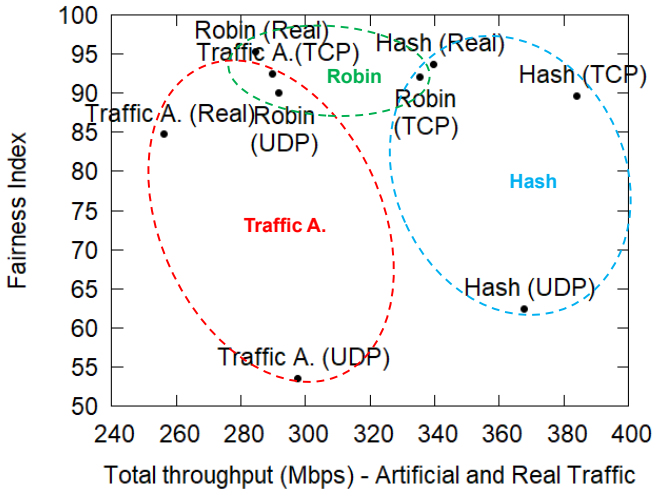

Figura 6.12. Gráfico de dispersão dos índices de justiça obtidas nos testes

interface (ou o mesmo conjunto de interfaces dentro da agregação) seja selecionada no início dos fluxos. De qualquer forma, uma análise mais aprofundada precisa ser realizada utilizando captura de pacotes, e com isso otimizações no código do algoritmo precisam ser feitas.

É definitivamente notável a alta precisão que a abordagem apresentada obteve no laboratório da vida real em comparação à utilização nativa do *Linux Bounding*. Os algoritmos Tabela *Hash* e *Round-Robin* virtual conseguiram dividir o tráfego igualmente entre todas as interfaces da agregação local, da mesma forma que o algoritmo nativo.

Por fim, o ótimo desempenho dos algoritmos Tabela *Hash* e *Round-Robin* virtual

novamente evidencia a necessidade de aprofundar os estudos de agregação de enlaces em redes SDN.

Capítulo 7

Conclusão e Trabalhos Futuros

Os resultados apresentados neste trabalho demonstram que existe a necessidade de pesquisar a utilização da técnica de agregação de enlaces em ambientes de redes SDN, uma vez que o modelo clássico (geralmente utilizando o protocolo LACP) pode ser melhorado. O dinamismo, a complexidade e a escalabilidade dos ambientes de redes atuais justificam a necessidade de encontrar uma nova solução para o antigo problema de gargalo, que é a principal contribuição deste trabalho.

A decisão de modificar o caminho dos pacotes periodicamente se apresentou como a melhor forma de balancear o tráfego entre todas as interfaces de uma agregação, considerando todas as propostas discutidas. Tal premissa fez com que a distribuição se tornasse mais justa e os resultados apresentados se aproximaram dos mesmos resultados obtidos utilizando uma solução amplamente usada em ambientes reais. Além disso, a solução apresentada foi capaz de utilizar 84% da banda disponível na transmissão de tráfego TCP e 91,4% no tráfego UDP. O índice de justiça também foi bastante satisfatório, ultrapassando 94% nos tráfegos TCP e 98% nos tráfegos UDP.

Além dos algoritmos propostos, este trabalho também contempla a detecção de mudanças na rede, fazendo com que a adição ou remoção de enlaces de uma agregação seja feita de forma automática. Vale lembrar que, na solução clássica, as novas interfaces devem ser inseridas manualmente na interface virtual criada pelo protocolo.

Como trabalhos futuros, a integração de um sistema com uma ferramenta de predição de tráfego para permitir engenharia de tráfego em tempo real está prevista. Além disso, a implementação completa da solução dinâmica e auto adaptável para o tráfego intra-fluxo precisa ser realizada. É necessário também estudar uma maneira de melhorar a implementação do algoritmo de Análise de Tráfego, já que a maneira como essa premissa foi implementada não trouxe resultados satisfatórios para a agregação de enlaces. A remoção de regras, mesmo sem a necessidade de esperar que elas expirem,

pode ser uma boa maneira de contornar a má distribuição do tráfego. O suporte ao protocolo MPTCP também é prevista. Por fim, o impacto da reescrita dos fluxos de tempos em tempos precisa ser avaliada, já que isso pode ser o causador de eventuais atrasos na transmissão de pacotes.

Referências Bibliográficas

Ahn, J. H.; Binkert, N.; Davis, A.; McLaren, M. & Schreiber, R. S. (2009). Hyperx: topology, routing, and packaging of efficient large-scale networks. Em *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pp. 1–11. ISSN 2167-4329.

Al-Fares, M.; Radhakrishnan, S.; Raghavan, B.; Huang, N. & Vahdat, A. (2010). Hedera: Dynamic flow scheduling for data center networks. Em *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pp. 19--19, Berkeley, CA, USA. USENIX Association.

Al-Shabibi, A.; Leenheer, M. D.; Gerola, M.; Koshibe, A.; Parulkar, G. M.; Salvadori, E. & Snow, B. (2014). Openvirtex: make your virtual sdns programmable. Em Akella, A. & Greenberg, A. G., editores, *HotSDN*, pp. 25–30. ACM.

Alizadeh, M.; Greenberg, A.; Maltz, D.; Padhye, J.; Patel, P.; Prabhakar, B.; Sengupta, S. & Sridharan, M. (2010). Dctcp: Efficient packet transport for the commoditized data center. *Digital Equipment Corporation. Tech. rep., Technical Report DEC-TR-301.*

Bredel, M.; Bozakov, Z.; Barczyk, A. & Newman, H. (2014). Flow-based load balancing in multipathed layer-2 networks using openflow and multipath-tcp. Em *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pp. 213--214, New York, NY, USA. ACM.

Carter, R. L. & Crovella, M. E. (1996). Measuring bottleneck link speed in packet-switched networks. *Performance evaluation*, 27:297--318.

Davis, T. D.; Tarreau, W.; Gavrilov, C.; N. Tindel, C.; Girouard, J. & Vosburgh, J. (2011). Linux ethernet bonding driver howto. *Digital Equipment Corporation. Tech. rep., Technical Report DEC-TR-301.*

Floyd, S.; Network Res. Group, Lawrence Berkeley Lab., C. U. & Jacobson, V. (1995). Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3:365--386.

Gude, N.; Koponen, T.; Pettit, J.; Pfaff, B.; Casado, M.; McKeown, N. & Shenker, S. (2008). Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105--110.

Guedes, D.; Vieira, L. F. M.; Vieira, M. A. M.; Rodrigues, H. & Nunes, R. V. (2012). Redes Definidas por Software: uma abordagem sistêmica para o desenvolvimento de pesquisas em Redes de Computadores. SBRC 2012.

IEEE (2014). Ieee standard for local and metropolitan area networks – link aggregation. *IEEE Std*.

Jain, R.; Chiu, D. & Hawe, W. (1984). A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. *DEC Research Report TR-301*.

Jain, R.; Chiu, D. & Hawe, W. (1984). A quantitative measure of fairness and discrimination for resource allocation in shared systems. *Digital Equipment Corporation. Tech. rep., Technical Report DEC-TR-301*.

Jouet, S. & Pezaros, D. P. (2017). Bpfabric: Data plane programmability for software defined networks. Em *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 38–48. ISSN .

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

Kim, H.; of Technology, G. I. & Feamster, N. (2013). Improving network management with software defined networking. *IEEE Communications Magazine*, 51:114--119.

Leiserson, C. E. (1985). Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901. ISSN 0018-9340.

Macedo, D. F.; Guedes, D.; Vieira, L. F. M.; Vieira, M. A. M. & Nogueira, M. (2015). Programmable networks: From software-defined radio to software-defined networking. *IEEE Communications Surveys and Tutorials*. ISSN 1553-877X.

McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S. & Turner, J. (2008). Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69--74.

McQuillan, J.; Beranek, B.; Newman Inc., M.; Richer, I. & Rosen, E. (1980). The new routing algorithm for the arpanet. *IEEE Transactions on Communications*, 28:711--719.

Moore, J. T. & Nettles, S. M. (2001a). Towards practical programmable packets. Em *in Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2001*, pp. 41--50.

Moore, J. T. & Nettles, S. M. (2001b). Towards practical programmable packets. Em *Proceedings of the 20th Conference on Computer Communications (INFOCOM). Citeseer*. Citeseer.

Morreale, P. A. & Anderson, J. M. (2014). *Software Defined Networking: Design and Deployment*. CRC Press.

Pfaff, B.; Pettit, J.; Amidon, K.; Casado, M.; Koponen, T. & Shenker, S. (2009). Extending networking into the virtualization layer. Em *Hotnets*.

Raiciu, C.; Barre, S.; Pluntke, C.; Greenhalgh, A.; Wischik, D. & Handley, M. (2011). Improving datacenter performance and robustness with multipath tcp. Em *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pp. 266--277, New York, NY, USA. ACM.

Ranum, M. J.; Landfield, K.; Stolarchuk, M.; Sienkiewicz, M.; Lambeth, A. & Wall, E. (1998). Implementing a generalized tool for network monitoring. *Information Security Technical Report*, 3:53—64.

Seaman, M. (1999). Link aggregation control protocol. *IEEE http://grouper. ieee. org/groups/802/3/ad/public/mar99/seaman*, 1:0399.

Steinbacher, M. & Bredel, M. (2015). Lacp meets openflow – seamless link aggregation to openflow networks. *TNC15 Networking Conference*.

Tennenhouse, D. L. & Wetherall, D. J. (2002). Towards an active network architecture. Em *DARPA Active Networks Conference and Exposition, 2002. Proceedings*, pp. 2--15. IEEE.

Tennenhouse, D. L. & Wetherall, D. J. (2007). Towards an active network architecture. *SIGCOMM Comput. Commun. Rev.*, 37(5):81--94.

Vencioneck, R. D.; Vassoler, G.; Martinello, M.; Ribeiro, M. R. & Marcondes, C. (2014a). Flexforward: Enabling an sdn manageable forwarding engine in open

vswitch. Em *10th International Conference on Network and Service Management (CNSM) and Workshop*, pp. 296--299. IEEE.

Vencioneck, R. D.; Vassoler, G.; Martinello, M.; Ribeiro, M. R. N. & Marcondes, C. (2014b). Flexforward: Enabling an sdn manageable forwarding engine in open vswitch. *10th International Conference on Network and Service Management (CNSM) and Workshop*, pp. 296--299.