TIAGO ANDRÉ CARBONARO DE OLIVEIRA

CLASSIFICAÇÃO DE REGRAS DE UM CONTROLADOR SDN UTILIZANDO REDES NEURAIS ARTIFICIAIS

LAVRAS – MG

2019

TIAGO ANDRÉ CARBONARO DE OLIVEIRA

CLASSIFICAÇÃO DE REGRAS DE UM CONTROLADOR SDN UTILIZANDO REDES NEURAIS ARTIFICIAIS

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Redes de Computadores e Sistemas Embarcados, para a obtenção do título de Mestre.

Prof. PhD Luiz Henrique Andrade Correia
Orientador

LAVRAS – MG

2019

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo próprio autor.

Oliveira, Tiago André Carbonaro de.
Classificação de regras de um controlador SDN utilizando Redes Neurais Artificiais / Tiago André Carbonaro de Oliveira. – 2019.
91 p. : il.

Orientador: Prof. PhD Luiz Henrique Andrade Correia.

Dissertação (mestrado acadêmico)–Universidade Federal de Lavras, 2019.
Bibliografia.

1. Redes Definidas por Software. 2. RNA. 3. *OpenFlow* I. Correia, Luiz Henrique Andrade. II. Título.

TIAGO ANDRÉ CARBONARO DE OLIVEIRA

CLASSIFICAÇÃO DE REGRAS DE UM CONTROLADOR SDN UTILIZANDO REDES NEURAIS ARTIFICIAIS

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Redes de Computadores e Sistemas Embarcados, para a obtenção do título de Mestre.

APROVADA em 27 de Fevereiro de 2019.

Prof. DSc. Neumar Costa Malheiros	UFLA
Prof. DSc. Alex Borges Vieira	UFJF

Prof. PhD Luiz Henrique Andrade Correia
Orientador

LAVRAS – MG
2019

RESUMO

O novo paradigma de Redes Definidas por Software – SDN (*Software Defined Networking*) estabelece a separação dos planos de controle e de dados. Neste tipo de rede, tal separação implica na inserção de mais um elemento da rede, o controlador. O protocolo de comunicação padrão conhecido como OpenFlow (OF), permite que os elementos de encaminhamento ofereçam uma interface de programação para o profissional de redes. Além disso, o OF possibilita ao administrador da rede, a extensão do acesso e controle da tabela de consulta utilizada pelo hardware. Esse modelo de controle determina o próximo passo de cada pacote, sendo construído através de regras programáveis. Diante desses novos conceitos e aplicações geradas pelas redes definidas por software, tornou-se necessário medir, testar e avaliar equipamentos que suportem esse padrão de tecnologia. O controlador atuando no switch SDN modifica diretamente o perfil do tráfego gerado, pois o mesmo vem acompanhado de influências de regras estáticas e dinâmicas (inseridas no controlador) que geram, em muitos casos, dúvidas ao profissional de rede sobre diagnósticos cotidianos. Uma ferramenta de classificação é muito útil nesse contexto de regras estáticas e dinâmicas, atuando sobre um tráfego amplo e repleto de valores variáveis. A utilização de uma Rede Neural Artificial (RNA), fornece excelentes resultados para a classificação de regras observando o contexto de funcionalidades de uma camada. Dessa forma, uma RNA foi proposta como modelo de classificação de regras do controlador SDN de acordo com os parâmetros de tráfego efetivos, gerados por duas ferramentas específicas o BWPING e o OSTINATO. Além disso, o tráfego gerado foi baseado em quatro tipos de protocolos: ICMP, TCP, UDP e HTTP. O controlador utilizado foi o POX e a função das regras foi aplicada sobre a camada de Enlace abrangendo aspectos de encaminhamento. Alguns parâmetros como *round-trip time*, atraso, largura de banda, quantidade de pacotes e regras da tabela de fluxo, serviram de base para a alimentação da RNA. A modelagem e a simulação foram realizadas por meio de um ambiente laboratorial com equipamentos de rede e, também, com a utilização do virtualizador Mininet para a SDN. Os resultados dos experimentos mostraram desempenho satisfatório da rede neural, atingindo cerca de noventa por cento de acerto.

Palavras-chave: SDN, RNA, OpenFlow, Controlador.

ABSTRACT

The new paradigm of Software Defined Networking (SDN) establishes the separation of control and data planes. In this type of network, such separation implies the insertion of another element of the network, the controller. The standard communication protocol known as OpenFlow (OF), allows routing elements to provide a programming interface for the network professional. In addition, OF allows the network administrator to extend the access and control of the query table used by the hardware. This control model determines the next step of each package, being built through programmable rules. Faced with these new concepts and applications generated by software-defined networks, it became necessary to measure, test and evaluate equipment that supports this technology standard. The controller acting on the SDN switch directly modifies the profile of the generated traffic, since it is accompanied by static and dynamic rules influences (inserted in the controller) that generate, in many cases, hesitations to the network professional about daily diagnoses. A classification tool is very useful in this context of static and dynamic rules, acting on a wide traffic and full of variable values. The use of an Artificial Neural Network (ANN), provides excellent results for the classification of rules observing the context of functionalities of a layer. Thus, an ANN was proposed as a classification model of rules of the SDN controller according to the effective traffic parameters, generated by two specific tools BWPING and OSTINATO. In addition, the traffic generated was based on four types of protocols: ICMP, TCP, UDP and HTTP. The controller used was the POX and the function of the rules was applied on the Link layer covering aspects of routing. Some parameters such as round-trip time, delay, bandwidth, number of packets, and flow table rules served as the basis for ANN feeding. The modeling and simulation were performed through a lab environment with network equipment and also using the Mininet virtualization for the SDN. The results of the experiments showed satisfactory performance of the neural network, reaching about ninety per cent accuracy.

Keywords: SDN, ANN, OpenFlow, Controller.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

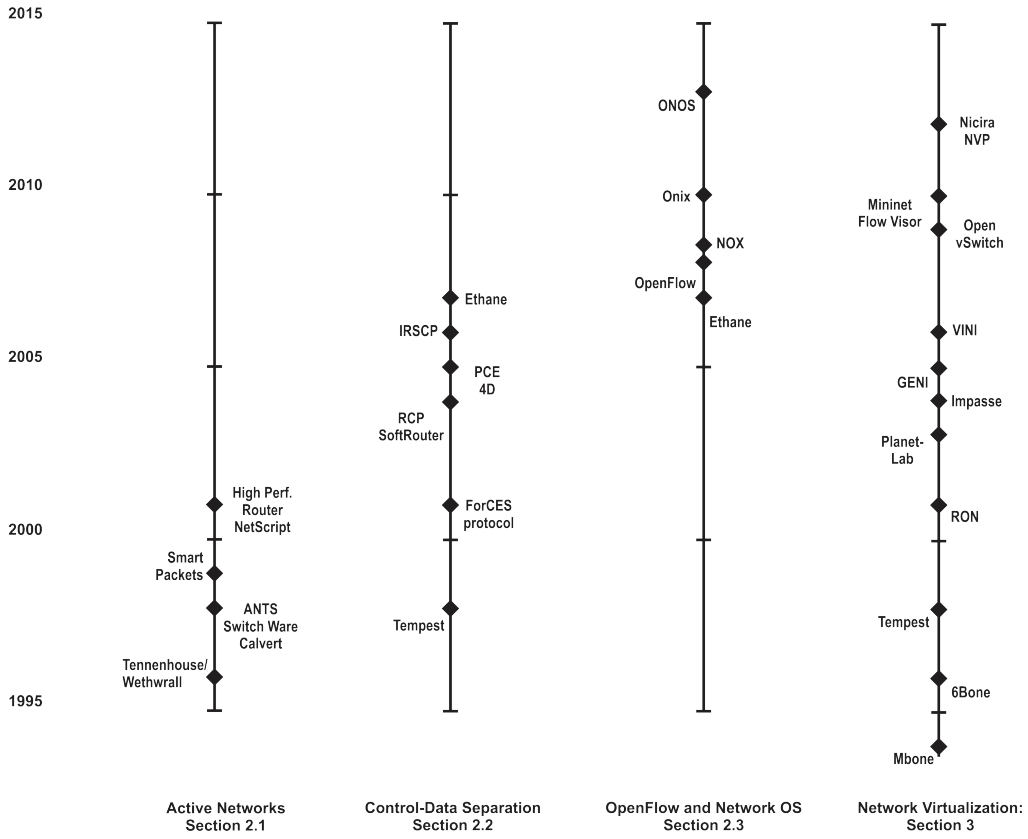
A infraestrutura das redes de pacotes é composta, atualmente, por equipamentos proprietários, fechados e de alto custo, cujas arquiteturas básicas são concebidas a partir da combinação de circuitos dedicados responsáveis por garantir alto desempenho (*Application-Specific Integrated Circuit*), ao processamento de pacotes (NASCIMENTO et al., 2011). Como consequência, as redes operam como um sistema de repasse de pacotes ou fluxos de dados genéricos. Assim, percebe-se que manipulações baseadas nas características das aplicações, como a alteração de dados e filtragem de informações, não são utilizados nas redes tradicionais.

As Redes Definidas por Software ou *Software Defined Network* - SDN visam resolver esses problemas citados anteriormente, como o alto custo dos equipamentos por exemplo, além disso, esse arquétipo de rede separa o controle do que é trafegado. Desse modo, o encaminhamento de pacotes pode ser configurado de forma mais amigável e melhor se adaptar às necessidades específicas de um ambiente de rede.

Agregado ao protocolo Openflow, em 2008, surgiram interfaces de programação para elementos de encaminhamento, o que possibilitou ao administrador da rede um acesso e controle das tabelas inseridas em ambientes antes manipulados exclusivamente através de hardware, podendo assim determinar o próximo passo de cada pacote recebido (GUEDES et al., 2012). Desse modo, a eficiência do encaminhamento se mantém, pois a consulta à tabela de encaminhamento continua sendo tarefa do hardware, mas a decisão sobre como cada pacote deve ser processado pode ser transferida para um nível superior, na qual diferentes funcionalidades podem ser implementadas. Tal estrutura permite que o controle da rede seja através de aplicações, expressas em software. Essa nova tecnologia foi chamada de Redes Definidas por Software.

Antes da criação das redes definidas por software (SDN), o objetivo era criar uma rede de computadores programáveis, permitindo a inovação no gerenciamento das redes e diminuindo a barreira, então existente, para criar novos serviços para a rede. Antes do surgimento das SDN, diversas outras tentativas de redes programáveis já haviam sido apresentas, como pode ser visto na Figura 1. Alguns dos projetos citados abaixo não vingaram no mercado, mas tiveram uma contribuição essencial ao projeto de desenvolvimento de redes programáveis.

Figura 1 – A evolução das redes programáveis



Fonte: adaptada de (FEAMSTER; REXFORD; ZEGURA, 2014)

O conceito e a aplicação da SDN, entretanto, expandiu muito desde então, incluindo uma variedade maior de tecnologias, aumentado significativamente o ramo de pesquisa na área.

Contudo, a associação entre SDN e Inteligência Computacional se torna um valioso campo de estudo no cenário contemporâneo de Redes de Computadores (MESTRES et al., 2017). Em especial no reconhecimento de padrões específicos de determinada aplicação e, consequentemente, na classificação dos mesmos. Como essas aplicações são oriundas de redes definidas e controladas por algum tipo de software externo ao hardware de encaminhamento de pacotes, e como tal controle é realizado através de regras inseridas no dispositivo controlador, classificar esse tipo regra torna-se um trunfo para o administrador da rede.

1.1 Motivação

As redes SDN surgiram com intuito de quebrar o sistema de verticalização e de códigos fechados oriundos de sistemas proprietários. A ideia central é trocar as antigas “caixas pretas” por sistemas programáveis e adaptáveis. Alguns benefícios com a utilização de SDN, são:

1. Separação dos planos de dados e de controle para tratar o fluxo de forma individualizada;

2. Aumentar o desempenho da rede em diversos aspectos como operação, monitoramento, controle de banda, latência e agregação de fluxo;

3. Reconfigurabilidade dos dispositivos de encaminhamento;

4. Testes a quente, proporcionando reconfiguração da rede e permitindo testes de experimentação;

5. Economia no quesito hardware específico de grande porte, como por exemplo suprir algumas funções de *firewalls* e ou roteadores.

A integração de SDN com outros métodos de decisão, baseados em inteligência computacional, pode proporcionar novas abstrações para as formas de encaminhamento e programação da rede (GUEDES et al., 2012).

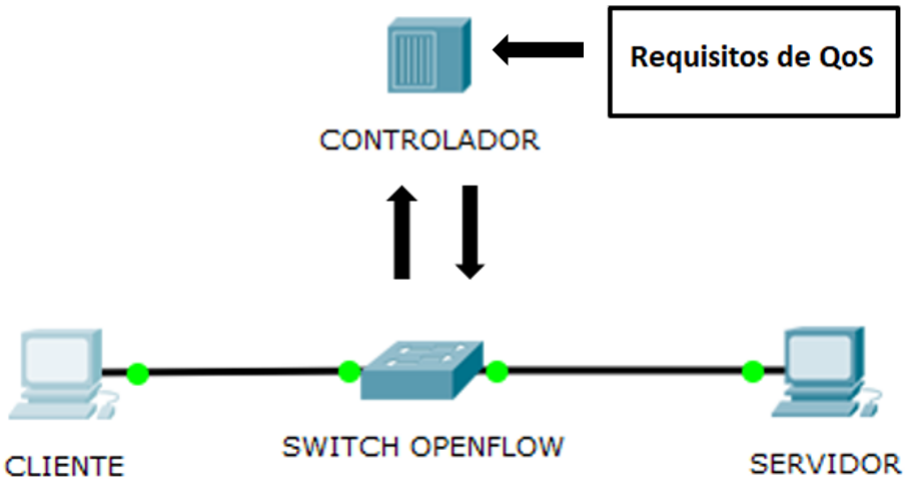
Mesmo conhecendo determinadas funções e características de cada regra (e parecer obvio a tomada de decisão) a utilização da inteligência computacional condiz para classificação automática em um ambiente com uma variação de topologias, protocolos e cargas de pacotes. Logo, a sugestão de classificação automática serve para facilitar a operação mantendo sempre um nível de excelência da prática operacional.

1.2 Definição do Problema

O crescimento significativo da utilização de SDN, a manipulação e a velocidade de resposta na operacionalização de tais redes passaram a ser um ponto crítico e passível de análise.

A Figura 2 mostra um cenário de definição do problema, no qual são definidas regras de encaminhamento para o fluxo de dados entre o cliente e o servidor.

Figura 2 – Cenário básico de SDN



Fonte: (PRÓPRIO AUTOR)

O problema é encontrar regras para o controlador que mantenham os requisitos predeterminados de Qualidade de Serviço (*Quality of Service* - QoS) da aplicação, considerando os diferentes fluxos e perfis de rede existentes, gerando um mapeamento com tais especificações de QoS de alto nível de abstração, para uma gama de ações de configuração dos equipamentos de rede. Com a regra correta aplicada, uma melhoria na performance de um fluxo com comunicação fim-a-fim acontece com consequência da aplicabilidade certeira de uma regra que possui melhor desempenho para determinado perfil da rede.

1.3 Objetivos

Este trabalho tem como objetivo principal classificar diferentes regras para controladores SDN, especificamente regras dinâmicas atuando em conjunto com um tipo de regra estática de camada dois sob o aspecto de encaminhamento de pacotes. No qual regras dinâmicas são aquelas que podem ser instaladas pelo controlador sem a necessidade de compatibilidade de um fluxo SDN, ou seja, um fluxo com campos pertencentes a determinado subconjunto. Eliminando assim o casamento conhecido como *match*, ou a compatibilidade de fluxo como dito anteriormente. As regras estáticas, por sua vez, são combinações de um ou mais campos de cabeçalho desenvolvidos, pelo administrador da rede, com intuito de cumprir uma ação específica, seja encaminhamento de pacotes ou até mesmo roteamento, por exemplo.

Outrossim, os experimentos buscam auxiliar no surgimento de possíveis novas ferramentas de inteligência computacional que contrastem com o modelo de inserção de regras tradicional. Modelo no qual uma regra dinâmica é inserida sem uma análise prévia do administrador da rede, ocasionando perda de performance da mesma. Os resultados mostram uma classificação consistente de três tipos de regras dinâmicas, atingindo mais de 90% de acertos nas fases de teste, treinamento e validação cumprindo, assim, com as especificidades da pesquisa.

Os objetivos específicos do trabalho estão divididos como segue:

1. Combinar os três conjuntos de regras dinâmicas de *layer 2* com dois tipos de regras estáticas, também de *layer 2*;
2. Combinar os diferentes tipos de topologia, protocolo e quantidade de pacotes com as regras;
3. Adaptar a operação de *flowstats* para captura da métrica de atraso;
4. Configurar e treinar a RNA por meio dos *traces* coletados e dos conjuntos de regras do controlador;
5. Realizar o pós-processamento dos atrasos gerados;
6. Transformar os protocolos, as topologias e a quantidade de pacotes em valores numéricos fixos;
7. Coletar os valores de atraso, topologia, protocolo e quantidade de pacotes para formação da tabela de entrada da RNA;
8. Realizar os mesmos testes com hardware-switch;
9. Comparar performance de um hardware-switch para com o softwares-switch;
10. Comparar performance do método tradicional de inserção de regras com o gerado pela RNA;
11. Medir o custo operacional gerado com a troca de regras no controlador.

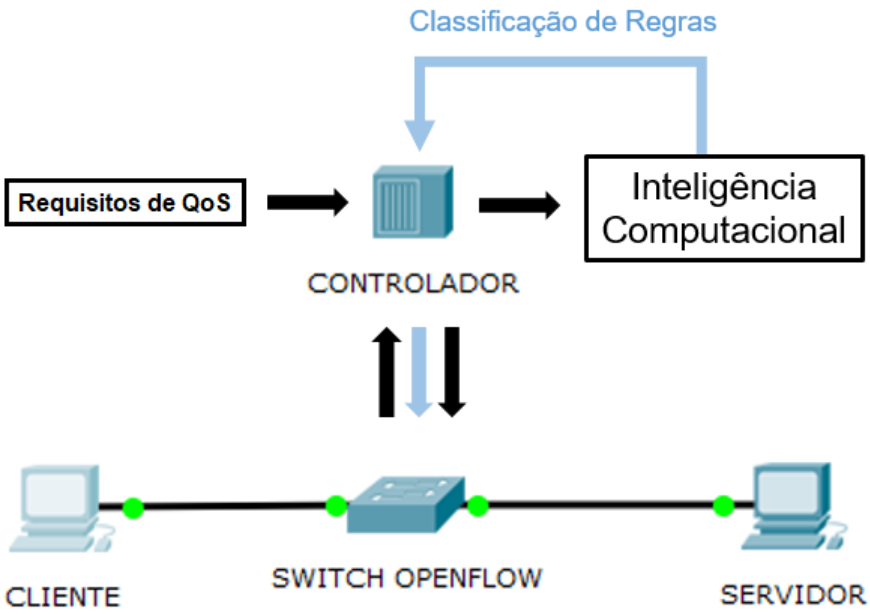
1.4 Solução Proposta

A utilização de ferramentas de apoio ao administrador de rede, com a presença de um controlador atuando diretamente nos dispositivos de encaminhamento, tornaram-se excelentes opções para melhoria de performance. Ademais, a integração de inteligência computacional com redes definidas por software pode suscitar respostas importantes para o segmento (QAZI et al., 2013).

Embora redes SDN serem muito utilizadas na indústria e citadas em inúmeras pesquisas acadêmicas, a literatura ainda é carente de material de pesquisa que unem SDN e inteligência computacional, especialmente na tomada de decisão de regras que atuam no controlador.

Desse modo, a inserção de um mecanismo inteligente, como mostrado na Figura 3, que classifique as regras para o controlador, de forma automática, e que mantenha os requisitos predeterminados para a rede (QoS), auxiliará na operação e administração das SDN, inclusive com melhoria de performance. Entretanto, um ponto importante é analisar os fluxos de entrada da rede além do seu perfil de variação nos quesitos de topologia, protocolos e cargas de pacotes, para indicarmos a forma de atuação de tal mecanismo.

Figura 3 – Cenário SDN com Solução do Problema



Fonte: (PRÓPRIO AUTOR)

A utilização de várias topologias laboratoriais, protocolos e carga de pacotes, a inserção de regras compatíveis e a inserção de métricas de medição de performance foram utilizadas como base para a realização dos experimentos. Destarte, tais experimentos geraram resulta-

dos que podem fornecer suporte as áreas de pesquisa e as redes corporativas inseridas neste contexto.

1.5 Estrutura do Trabalho

Este trabalho está dividido em cinco seções. A próxima Seção apresenta o referencial teórico e os trabalhos relacionados. A Seção 3 apresenta a metodologia e materiais necessários para desenvolvimento da pesquisa. Os resultados obtidos são apresentados na seção 4. Por fim, na Seção 5, são apresentadas as conclusões e trabalhos futuros.

2 REFERENCIAL TEÓRICO

Esta seção explana sobre os principais conceitos, sistemas e valores que são pré-requisito para o entendimento das técnicas aplicadas nesse trabalho. Inicialmente serão fornecidos dados pertinentes às redes definidas por software e seus principais componentes, em seguida informações do controlador e virtualização de redes. Também serão mencionados dados sobre o virtualizador de rede, além dos geradores de tráfego utilizados na pesquisa. Na subseção seguinte as redes neurais artificiais serão explanadas e, por fim, os trabalhos relacionados serão abordados.

2.1 Redes Definidas por Software

As redes SDN possuem características específicas. Entre elas está a utilização de um software capaz de controlar o encaminhamento de pacotes pela rede, por intermédio de um conjunto de códigos de programação bem definidos, o qual inspeciona, define, gerencia e altera entradas dentro da parte inteligente de cada equipamento – contando, para isso, com a colaboração de outro software gerenciador.

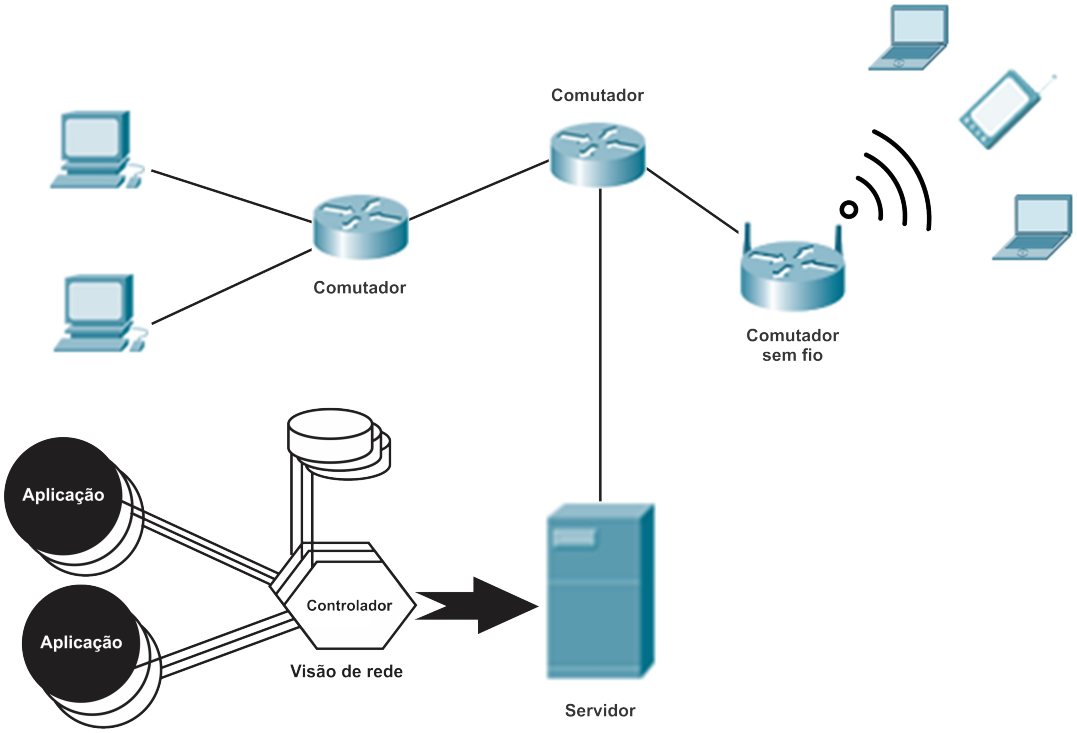
Além do mais, as Redes Definidas por Software fogem da arquitetura estática das redes tradicionais, que não fornecem suporte às necessidades de computação e de armazenamento dinâmicos e escaláveis de ambientes de computação mais modernos, como *Data Centers*. Tal processo é feito desassociando o sistema que toma decisões relacionadas ao para onde o tráfego é enviado (o plano de controle) dos sistemas subjacentes, que encaminham o tráfego para o destino selecionado (o plano de dados).

Comparadas às redes tradicionais, Redes Definidas por Software possuem características e benefícios (JUNIPER, 2017). A seguir algumas das principais características e benefícios:

- Separação em três camadas: gerenciamento, controle e encaminhamento;
- Centralização de aspectos relevantes dos planos de gerenciamento, serviços e controle: simplifica o projeto da rede e reduz os custos operacionais;
- Criação de uma plataforma para aplicações de rede, de serviços e de integração com sistemas de gerenciamento: permite novas soluções de negócios;
- Padronização de protocolos: propicia a interoperabilidade e o suporte heterogêneo de fornecedores com mais opções e redução de custos.

A Figura 4 mostra a estrutura de uma SDN de modo simplificado. A tabela de encaminhamento do computador é consultada a cada pacote recebido pelo switch, por exemplo. Se nada constar na referida tabela, ou não existir nenhuma ação definida para o pacote, o switch encaminha para o controlador. Desse modo, o controlador insere uma nova regra à tabela de encaminhamento do switch, para evitar que a rotina seja repetida no caso de uma futura ação de pacotes do mesmo tipo (GUEDES et al., 2012).

Figura 4 – Estrutura geral de uma SDN



Fonte: adaptada de (GUEDES et al., 2012)

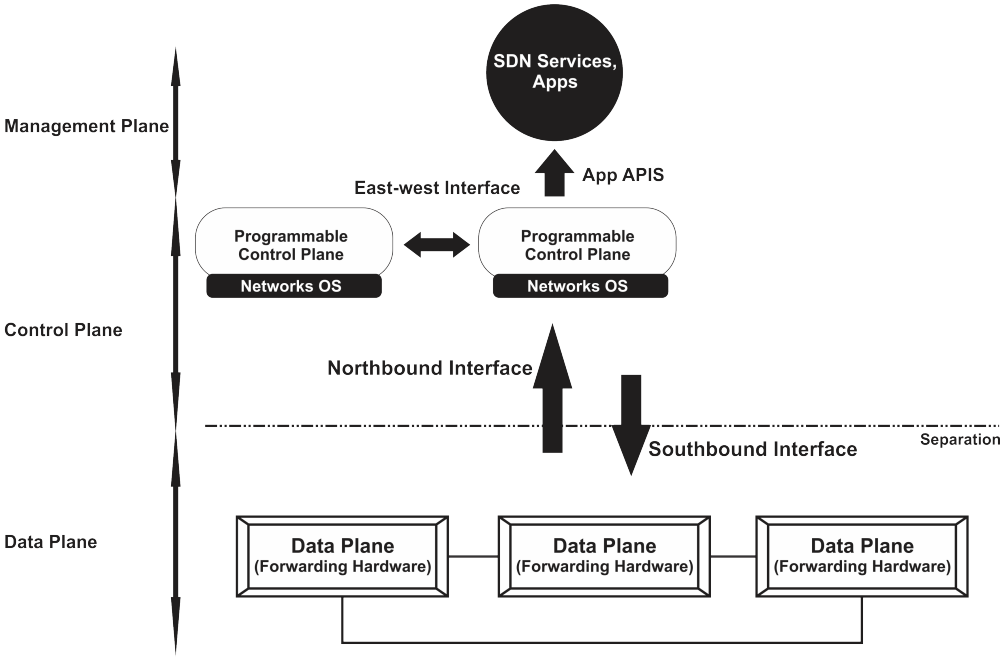
2.2 Arquitetura das Redes Definidas por Software

A arquitetura de Redes Definidas por Software pode ser dividida em três partes principais: plano de gerenciamento, plano de controle e plano de dados. Além disso, o plano de controle possui uma interface para o sul (*Southbound Interface*) (comunicando com o plano de dados) uma interface para o norte (*Northbound Interface*) (comunicando com o plano de gerenciamento).

A Figura 5 mostra, de forma resumida, a arquitetura de uma SDN. Existe uma linha de separação, ente os planos de dados e de controle, que reflete uma abordagem onde o controle e gerenciamento estão localizados dentro de um conjunto (com as suas funcionalidades dividas

em dois subconjuntos) e a parte de dados está localizada dentro de outro conjunto (SHIN; NAM; KIM, 2012).

Figura 5 – Arquitetura de referência SDN e APIs



Fonte: adaptada de (SHIN; NAM; KIM, 2012)

As subseções a seguir irão detalhar cada um dos componentes arquitetônicos supracitados, em uma abordagem *TOP - DOWN*.

2.2.1 Plano de Gerenciamento

O plano de gerenciamento é a interface entre a SDN e o administrador da rede. Nesse ambiente é possível programar os equipamentos da rede para desempenhar várias funções, além da possibilidade de obtenção de informações sobre o estado da rede e do recebimento de notificações dos eventos. Essas funções são realizadas de forma facilitada devido às APIs, (*Application Programming Interfaces*) que realizam a comunicação entre o plano de gerenciamento e o controlador. Para mais, as linguagens de programação usadas nas aplicações desse plano fornecem várias abstrações, facilitando, assim, a programação das aplicações e a reutilização de códigos.

As principais aplicações de rede em uma SDN exercem as funções similares as das redes de computadores convencionais, como roteamento, balanceamento de carga e aplicação de políticas de segurança. Além disso, algumas funções mais recentes são também realizadas, tais como economia de energia, aplicação de Qualidade de Serviço fim-a-fim, virtualização de

redes, gerenciamento de mobilidade em redes sem fio, engenharia de tráfego. A variedade de aplicações de rede, que podem ser desenvolvidas com rapidez e com maior facilidade por meio das APIs e das linguagens de programação, é um dos argumentos mais fortes para a adoção de uma SDN (GOMES et al., 2013).

2.2.2 Northbound Interface

A *Northbound Interface* é uma peça chave da arquitetura da SDN. Ela é uma API que realiza a comunicação entre o plano de gerenciamento e o plano de controle efetivamente, permitindo, assim, que aplicações utilizadas por administradores de rede efetuem o controle e o monitoramento das funções da rede sem ter que ajustar os detalhes mais finos da comunicação (KREUTZ et al., 2015). Isso é possível também devido à *Southbound Interface*, descrita mais adiante.

O ideal para esse tipo de interface é o estabelecimento de um padrão, de modo que se gere uma abstração que independa da linguagem de programação e do controlador. No entanto, isso ainda não foi definido. Cada controlador especifica sua própria API. A principal função da *Northbound Interface* é converter os requisitos das aplicações de gerenciamento em instruções de baixo nível e transmitir estatísticas, geradas nos dispositivos da rede e processadas pelo controlador, sobre a rede.

2.2.3 Plano de Controle

O Plano de Controle é responsável por processar e delegar as operações inteligentes, definidas em software, e inseridas nos seus componentes. Tais componentes são o sistema operacional da rede (*Network Operating System* - NOS), que oferece um controle lógico e o próprio controlador. O NOS, como qualquer outro sistema operacional, fornece abstrações, serviços essenciais e APIs para desenvolvedores. Isso significa que um desenvolvedor não precisa mais saber de todos os detalhes de uma transmissão de pacotes para definir uma política de rede, situação que facilita o seu desenvolvimento e diminui a chance de erros. Geralmente, ele pode ser instalado em um computador de porte médio.

As principais funções de um NOS são: análise do estado da rede, fornecimento de informações sobre a topologia, descoberta de dispositivos conectados à rede, distribuição de configurações da rede, gerenciamento de dispositivos, encaminhamento de dados pelo caminho mais curto, mecanismos de segurança, recebimento, processamento e encaminhamento de eventos.

Recursos de segurança são de primordial importância, pois eles possibilitam isolamento e aplicação de regras entre serviços e aplicações.

Um NOS deve fornecer uma interface comum para as camadas superiores, enquanto permite que uma plataforma de controle use diferentes Southbound APIs e plug-ins de protocolos, permitindo, assim, além de gerenciar diferentes dispositivos da rede, oferecer compatibilidade entre diferentes versões.

Existem controladores centralizados e descentralizados. Os centralizados possuem a desvantagem de utilizar apenas um único controlador da rede, aglutinando todas as funções em único equipamento. Além disso, um único controlador pode não ser o suficiente no que se refere a gerenciar uma rede com muitos elementos no plano de dados.

Já o uso de controladores distribuídos pode ser escalado para atender aos requisitos de qualquer ambiente (KREUTZ et al., 2015). Eles caracterizam-se por um conjunto de nós concentrados em uma localidade ou por vários elementos de processamento distantes uns dos outros. A vantagem de um controlador distribuído é a maior adaptabilidade para com falhas físicas e lógicas (CHEN et al., 2017). Conquanto, um obstáculo encontra-se em manter o estado da rede sempre atualizado para todos os componentes do controlador.

Para a comunicação entre os nós do controlador, são usadas APIs chamadas de *Westbound* API e *Eastbound* API. Algumas funções dessas APIs são transmissão de dados entre controladores, algoritmos para a manutenção da consistência dos dados e de recursos de monitoramento e de notificação. Cada controlador implementa suas próprias APIs desse tipo.

2.2.4 Southbound Interface

A *Southbound Interface* é a ponte entre os elementos de controle e o encaminhamento de dados e é também considerada uma API. Esse é o elemento vital para a separação entre os planos de controle e dados. É por meio dela que os controladores da SDN podem comunicar à rede os requisitos das aplicações, reprogramando os equipamentos para que eles desempenhem inúmeras funções, como controle de fluxo, *firewall*, sistemas de detecção de intrusos (*Intruder Detection System* - IDS), além de roteamento e de comutação (KREUTZ et al., 2015). Essa reprogramação é feita adicionando ou removendo regras das tabelas de fluxo, as quais serão apresentadas com detalhes na descrição sobre o plano de dados.

Além disso, essa API é utilizada para fazer a comunicação entre os equipamentos de rede e o controlador. Tal comunicação acontece quando ocorre:

1. Envio de avisos de eventos caso ocorra uma mudança de porta ou enlace;

2. Envio de estatísticas de fluxo para o controlador, de forma a fornecer informações mais detalhadas sobre as características da rede para administradores da própria rede;

3. Envio de pacotes para o controlador (em dois casos): se os equipamentos do plano de dados não souberem o que fazer com um pacote, por não haver uma regra definida, ou quando alguma das regras instaladas no equipamento tem como comando "enviar para o controlador".

Todos esses tipos de comunicação estão definidos na *Southbound Interface* mais utilizada, o OpenFlow. No entanto, existem outras APIs, como OVSDB, ForCES e OpFlex, as quais adicionam novas funções (PFAFF et al., 2009). Por exemplo, a interface OVSDB permite aos elementos de controle a criação de várias instâncias de comutadores virtuais e configuração de políticas de Qualidade de Serviço nas interfaces. Por isso, ela é uma interface complementar ao OpenFlow para o uso com *Open vSwitch* (OVS, 2018).

2.2.5 Plano de Dados

O função do referido plano é encaminhar os dados na rede. É composto de dispositivos de encaminhamento, que são elementos de hardware ou de software especializados em encaminhar pacotes. Porém, diferentemente de roteadores e comutadores convencionais, eles não carregam implementações de protocolos e programas complexos. Em um dispositivo configurado para OpenFlow, por exemplo, apenas estão instaladas tabelas de fluxos, que decidem quais as ações que serão tomadas com cada pacote. Elas são compostas por regras de compatibilidade de fluxos, isto é, regras de comparação, que verificam valores de campos do cabeçalho do pacote, como o destino, o protocolo e o número de porta com algum valor esperado; e ações a serem tomadas com o pacote caso a regra valha. As ações que podem ser realizadas são:

- Encaminhar o pacote para uma porta de saída;

- Encapsular e encaminhar o pacote para o controlador;

- Descartar o pacote;

- Enviar o pacote para o pipeline de processamento;

- Enviar o pacote para a próxima tabela de fluxos ou para alguma tabela especial. A possibilidade de utilização de uma sequência de tabelas de fluxo ou do uso de mais de uma tabela está diretamente relacionada à versão apropriada do protocolo OpenFlow, ou seja, nem todas as versões são compatíveis.

Caso nenhuma regra seja satisfeita, o pacote pode ser descartado, mas o mais comum é que seja criada uma regra padrão (*default*), que encaminhe para o controlador qualquer pacote que não satisfaça nenhuma das outras regras. A prioridade entre as regras segue o ordenamento das linhas de uma tabela e, posteriormente, a ordem das tabelas em uma sequência de processamento. A Figura 6 ilustra os campos de uma tabela de fluxos além de exemplificar esse tipo de tabela.

Figura 6 – Exemplo de tabela de fluxos. Abaixo, os campos do cabeçalho usados na especificação de fluxos do OpenFlow

Tabela de Fluxos			
Número da Regra	**Regra**	**Ação**	**Contador**
1	IP de destino 192.168.0.0/24	Encaminhar para o controlador	40
2	Pacotes com interface de entrada 2 e VLAN ID 10	Bloquear	550
3	Porta TCP de destino 80	Encaminhar pela porta 400	0

Porta de Entrada	Endereço MAC de Origem	Endereço MAC de Destino	EtherType	ID da VLAN	Prioridade da VLAN
Endereço IP de Origem	Endereço IP de Destino	IP ToS	Protocolo IP	Porta UDP/TCP de Origem	Porta UDP/TCP de Destino

Fonte: adaptada de (COSTA et al., 2016)

2.3 Controladores

Os controladores oferecem uma visão centralizada da rede global e permitem que os administradores de rede ditem aos sistemas subjacentes (switches e roteadores) como o plano de encaminhamento deve lidar com o tráfego da rede.

O gerenciamento de redes em baixo nível é configurado em cada nó da rede, o que torna o serviço um dos maiores problemas na Internet. Em grande parte dos casos , as configurações são realizadas manualmente e são susceptíveis a falhas de configuração dos componentes (MACAPUNA et al., 2011). Para tratar problemas do gênero, implementou-se um sistema operacional de redes que oferece uma plataforma de desenvolvimento que permite a reutilização de aplicações de rede e possui a capacidade de análise e gerenciamento, permitindo assim que

aplicações de gerência / controle de rede sejam rapidamente desenvolvidas e aplicadas (GUDE et al., 2008).

O controlador de rede, ou sistema operacional de rede, ou ainda, *hypervisor* da rede (em alusão ao conceito derivado da área de sistemas virtualizados), pode concentrar a comunicação com todos os elementos programáveis que compõem a rede e oferecer uma visão unificada do estado da rede (SHERWOOD et al., 2009).

Uma das vantagens das Redes Definidas por Software, é exatamente essa visão de controlador. É possível gerenciar um sistema inteiro através de um único centralizador, com visão das condições da rede, o que possibilita tomar decisões operacionais sobre como um sistema todo deve operar.

Existem inúmeros sistemas operacionais de rede atuando nos controladores, os mais utilizados atualmente são o POX, o *Floodlight*, o ONOS e o *OpenDaylight* (SHERWOOD et al., 2009). Nas subseções seguintes, serão apresentados os tipos de controladores relacionados ao trabalho.

2.3.1 NOX

O sistema operacional de rede ou controlador NOX é original do OpenFlow, e tem como principal função o desenvolvimento de programas SDN na linguagem C++. O NOX funciona, assim como todos os controladores, sobre o conceito de fluxos de dados. O controlador gerencia a tabela de fluxos dos switches da rede reagindo a eventos de rede (GUEDES et al., 2012; NOX, 2018).

Outra característica desse controlador é definir um ação para cada um dos eventos construídos, isso por regra em execução. Podendo ser estruturado como um programa arbitrário. O NOX utiliza essa premissa como base de execução de todas as suas ações.

2.3.2 POX

O sistema operacional de rede POX foi implementado baseado no controlador NOX. É uma plataforma para o desenvolvimento e prototipagem de software de rede utilizando a linguagem *Python* (GUEDES et al., 2012).

O controlador POX surgiu com o objetivo de facilitar e contribuir em pesquisas e ensino sobre o protocolo OpenFlow. O POX disponibiliza uma interface mais simples e uma pilha SDN mais organizada, o que facilita a execução de pesquisas e estudo (COSTA, 2013). O principal

objetivo do POX é substituir o NOX em situações em que o desempenho não é algo para se preocupar (GUEDES et al., 2012; POX, 2018).

2.3.3 Comparativo entre o NOX e o POX

Devido a difícil instalação e manutenção, problemas na compilação e número significativo de dependências na montagem, variações do NOX surgiram: como NOX com *Python* e o POX.

O repositório do NOX (NOX, 2018) submeteu as duas versões do controlador NOX (com as interfaces de programação C++ e *Python*) e o controlador POX a uma comparação de desempenho baseadas em latência e taxa de transferência. Seguem os resultados com a métrica de latência em milissegundos:

- O NOX com o *Python* obteve uma resposta de 0,18 ms;
- O POX obteve uma resposta de 0,06 ms;
- E o NOX com o C++ obteve uma resposta de 0,02 ms.

A seguir os resultados da taxa de transferência em fluxos por segundo:

- O NOX com o C++ obteve uma resposta de 50 fluxos por segundo;
- O POX obteve uma resposta de 31 fluxos por segundo;
- E o NOX com o *Python* obteve uma resposta de 5 fluxos por segundo.

2.3.4 OpenDaylight

É um sistema que oferece uma implementação direta de SDN sem a necessidade de outros componentes, de forma efetivamente funcional (SUJITHA; MANIKANDAN; ASHWINI, 2018). Esse sistema operacional de rede oferece um *Framework* Java com o intuito de acelerar a adoção das SDNs, suporta OpenFlow assim como outros padrões de SDN, como também fornecer um controle para a rede que expõe APIs *northbound* abertas.

2.4 Função de Virtualização de Redes

As Redes Definidas por Software abriram espaço para que pesquisadores e desenvolvedores possam usar a rede como um ambiente de testes (GUEDES et al., 2012). Todavia, ao se conectar os elementos de comutação de uma rede a um controlador único, a capacidade de se desenvolver novas aplicações e testá-las, fica restrita ao responsável por aquele controlador. Mesmo que o acesso a este controlador seja compartilhado entre os pesquisadores, ainda há a questão da garantia de não interferência entre as diversas aplicações sendo executadas na mesma interface.

Para se resolver esse problema, surgiu a possibilidade de dividir a rede em fatias (*slices*) e atribuir cada fatia a um controlador diferente. Ao considerar que um controlador SDN é um sistema operacional de rede, uma forma de implementar essas fatias é através da virtualização.

A maior vantagem das Redes Definidas por Software são as diversas formas de se dividir os recursos encontrados na rede (MCKEOWN et al., 2008). A capacidade que o OpenFlow possui de permitir a divisão dos comportamentos entre o ambiente de produção e de pesquisa é bastante útil para o avanço das arquiteturas hoje existentes.

A capacidade de dividir a rede em fatias já ocorre na arquitetura atual da Internet, um exemplo é a utilização de VLANs, em que há um cabeçalho exclusivo no pacote para a definição de qual rede virtual ele pertence. Porém, o uso de VLANs possui limitações definidas pela tecnologia Ethernet, e por esse motivo torna-se complexo sua aplicação em contextos nos quais essas fatias devam se estender por mais de uma tecnologia de rede.

Considerando essa analogia do uso de VLANs com o contexto atual das redes SDN é possível estender essa divisão de forma que os recursos da rede sejam virtualizados e apresentados de maneira isolada para que cada desenvolvedor possa ter o seu próprio controlador de rede (COSTA, 2013).

Dessa forma, é permitido a integração do ambiente de produção com vários tipos de pesquisas, de forma paralela, na mesma rede física. A implementação de diferentes tipos de controladores também é suportada, permitindo que o desenvolvedor faça a análise do desempenho de cada controlador com sua respectiva linguagem de programação. Do mesmo modo que um sistema operacional é virtualizado, pode-se efetuar a virtualização de controladores sobre uma rede física.

2.5 Virtualizador de rede Mininet

Virtualizadores ou simuladores de ambientes de computacionais, em especial na área de redes, são facilitadores para montagem de cenários e ambientes para testes bem completos, tais ambientes fornecem subsídio para criação de leiautes sem custo com equipamentos físicos, por exemplo. O Mininet possui um ambiente utilizado para o desenvolvimento de aplicações e realização dos diversos testes. Trata-se de um emulador de redes, desenvolvido por pesquisadores da Universidade de *Stanford* nos Estados Unidos, cujo objetivo é apoiar pesquisas colaborativas permitindo protótipos autônomos de redes definidas por software, para que qualquer pessoa possa fazer o download, executar, avaliar, explorar e ajustar (LANTZ; HELLER; MCKEOWN, 2010).

O Mininet é um emulador do tipo CBE (*Container-Based Emulation*) que emprega a virtualização a nível de processo, uma forma mais leve de virtualização, o qual muitos recursos do sistema são compartilhados. Compartilhando recursos como tabela de paginação, estruturas de dados do *kernel* e sistema de arquivos, esse tipo de emulação alcança maior escalabilidade que outras que realizam uma emulação completa, ou seja, usando uma máquina virtual para cada dispositivo da rede (LANTZ; HELLER; MCKEOWN, 2010).

Tal vitualizador é capaz de emular links, hosts, switches e controladores, utilizando processos que são executados em espaços de nomes da rede (*network namespaces*) e pares *Ethernet* virtuais. Os itens mencionados serão resumidos dentro do contexto do Mininet.

- Links: um par Ethernet virtual atua como um cabo conectando duas interfaces virtuais. Pacotes enviados através de uma interface são entregues na outra e cada interface se comporta como uma interface Ethernet completa e funcional para todo o sistema e aplicação.

- Host: é simplesmente um processo do *shell* movido para o seu próprio espaço de nome da rede, ou seja, cada host apresenta uma instância da interface de rede independente. Cada dispositivo apresenta uma ou mais interfaces virtuais e um *pipe* para um processo pai do Mininet (mn), que envia comandos e monitora a saída.

- Switches: switches OpenFlow são remotamente configurados e gerenciados pelo controlador, podendo ser configurados para agirem como switches de *layer* 2 ou *legacy*.

- Controladores: controladores OpenFlow podem estar em qualquer lugar da rede, ou seja, tanto na rede física quanto na virtual (LANTZ; HELLER; MCKEOWN, 2010).

Para controlar e gerenciar todos os dispositivos emulados, o Mininet fornece uma CLI (*Command Line Interface*), ou seja, através de um único console, pode-se controlar todos os dispositivos emulados. Outra grande vantagem desse emulador é a facilidade de criação de topologias customizadas através de uma API (*Application Programming Interface*) para programação em *Python*.

O projeto Mininet nasceu com o objetivo de facilitar os experimentos no campo de redes de computadores, principalmente para auxiliar nas pesquisas das redes definidas por software e do OpenFlow. O emulador Mininet encontra-se em constante evolução, sendo mantido por uma ativa comunidade online (MININET, 2018).

2.6 OSTINATO – Gerador e analisador de tráfego de rede

O OSTINATO é uma plataforma *open source* capaz de criar pacotes, além de gerar e analisar o tráfego de rede com uma GUI amigável. Possui uma API, construída em *Python*, para automação de testes de rede. Permite enviar pacotes de vários fluxos com diferentes protocolos em taxas diferentes (OSTINATO, 2018).

Outra característica do Ostinato é fornecer um gerador de tráfego e ferramenta de teste de rede para cada profissional de rede e desenvolvedor, algo que é mais complexo de ser realizado com equipamentos de teste de rede comercial existentes.

No contexto da pesquisa, tal ferramenta foi importante no suporte aos experimentos, projetando resultados com uma gama maior de protocolos.

2.7 BWPING

O BWPING (ALVES et al., 2018) é uma plataforma *open source* capaz de criar pacotes, bem como gerar e analisar o tráfego de rede com uma GUI amigável. Permite enviar pacotes de vários fluxos com diferentes protocolos em taxas diferentes.

Outra característica do BWPING é fornecer um gerador de tráfego UDP independente para cada profissional de rede e desenvolvedor, algo que pode ser um complicador com equipamentos de teste de rede comercial existentes.

No contexto deste trabalho, tal ferramenta forneceu suporte aos experimentos e gerou valores de resposta importantes, porém restritos no aspecto de variação de protocolos. No quesito entrega da métrica de atraso médio já formatada, e pronta para ser tabulada, foi de

muita valia. Porém, por gerar tráfego apenas com o protocolo UDP, foi substituída pelo gerador de tráfego OSTINATO.

2.8 Redes Neurais Artificias – RNA

A definição de regras para os controladores em SDN pode levar a novas formas de interação entre os componentes da rede. Essa definição de regras pode ser realizada por meio de algoritmos de inteligência computacional, como por exemplo as redes neurais artificiais - RNA.

As redes neurais são sistemas de computação adaptativos inspirados nas características de processamento de informação encontradas nos neurônios reais e nas características de suas interconexões (BISHOP, 1995).

O estudo de RNAs é fundamentado no comportamento das redes neurais biológicas. O cérebro humano possui uma enorme capacidade de processar informações, como por exemplo, o reconhecimento de fala e a segmentação de imagens. E, a partir desta análise, pesquisas são realizadas com a finalidade de reproduzir as reações do cérebro em máquinas.

A formação básica de uma RNA, consiste num conjunto de neurônios artificiais que interagem entre si, semelhantes ao funcionamento dos neurônios biológicos (SILVA; SPATTI; FLAUZINO, 2010). Basicamente, são sistemas paralelos de computação e suas implementações podem ser em hardware (realiza uma determinada tarefa a partir de componentes eletrônicos) ou em software (simulações por programação em um computador digital).

Uma rede neural possui sua capacidade de aprendizado baseada na extração e armazenamento de dados. O processo de aprendizagem é feito a partir de algoritmos, no qual os pesos sinápticos da rede são modificados de forma ordenada para alcançar o resultado desejado. Dada a informação coletada em pares de entradas e saídas, as redes neurais devem “aprender” e generalizar o conhecimento contido nas amostras, de modo que responda de forma adequada quando entradas não presentes no conjunto de treinamento forem apresentadas à rede.

Isto é possível devido ao fato de que a formulação é apoiada numa representação de uma rede real de neurônios. Dessa forma, problemas relacionados a reconhecimento de padrões podem ser solucionados através da implementação e uso adequado desta técnica. Algumas características importantes em uma rede neural artificial são:

- Robustez e tolerância a falhas: a eliminação de alguns neurônios não afeta substancialmente o seu desempenho global.

- Flexibilidade: pode ser ajustada a novos ambientes por meio de um processo de aprendizagem, sendo capaz de aprender novas ações com base na informação contida nos dados de treinamento.

- Processamento de informação incerta: mesmo que a informação fornecida esteja incompleta, afetada por ruído, ainda é possível obter-se um raciocínio correto.

- Paralelismo: um imenso número de neurônios está ativo ao mesmo tempo. Não existe a restrição de um processador que obrigatoriamente trabalhe uma instrução após outra.

A estrutura de uma rede neural é distribuída e possui habilidade de aprender. Portanto, sua utilização oferece alguns benefícios como a capacidade de realizar mapeamentos não-lineares entre entrada e saída; a adaptabilidade, mediante a um algoritmo de aprendizado; a tolerância a falhas e a analogia neurobiológica.

2.8.1 Neurônio Artificial

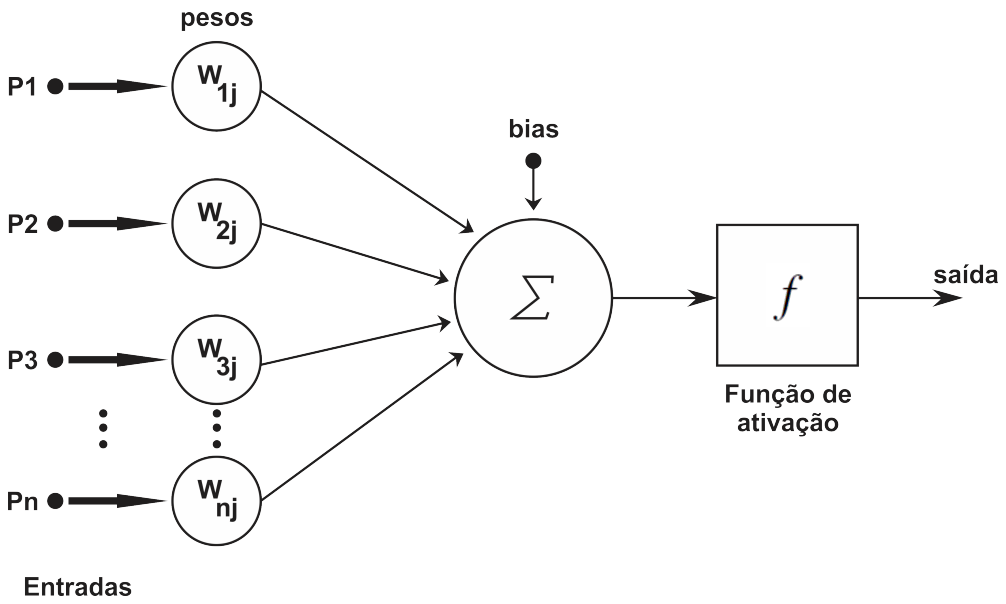
Todos os tipos de redes neurais apresentam o mesmo padrão, isto é, um neurônio artificial, que simula o comportamento do neurônio biológico. Tal neurônio artificial possui várias entradas, que correspondem às conexões sinápticas e uma saída, cujo valor é gerado por uma somatória ponderada de todas as saídas dos outros neurônios a esse conectado.

O modelo artificial de neurônio é mostrado na Figura 7, sendo uma generalização do modelo de *McCulloch* e *Pitts* (MCCULLOCH; PITTS, 1943). Esse modelo inclui um sinal adicional bias que favorece ou limita a possibilidade de ativação do neurônio. O processo sináptico é representado pelos pesos que amplificam cada um dos sinais recebidos. A chamada função de ativação modela a forma como o neurônio responde ao nível de excitação, limitando e definindo a saída da rede neural.

A função de ativação pode possuir diferentes representações. Os três tipos básicos de função de ativação são: limiar, linear e sigmoide. A função de ativação limiar seria um classificador alicerçado em limiar (*threshold*), por exemplo se um determinado valor estiver acima de um limite determinado, ative o neurônio senão desative o mesmo. A função de ativação linear utiliza uma função linear substituindo uma função de passo simples para correção dos pesos e adaptação do aprendizado, é uma função ideal para tarefas simples, na qual a interpretabilidade é altamente desejada. E a função de ativação sigmoide é apropriada para problemas não line-

ares. A escolha do tipo de função varia de acordo com os objetivos e particularidades de cada projeto (HAYKIN; NETWORK, 2004).

Figura 7 – Modelo artificial de neurônio biológico



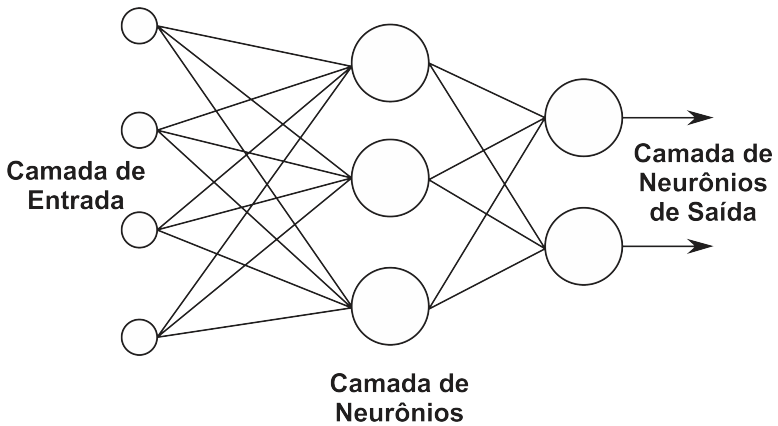
Fonte: adaptada de (MCCULLOCH; PITTS, 1943)

2.8.2 Redes Perceptron Multicamadas - MLP

Os perceptrons multicamadas possuem um tipo de arquitetura que se distingue pela presença de uma ou mais camadas ocultas ou intermediárias, cujos nós computacionais são chamados de neurônios ocultos ou unidades ocultas. A função dos neurônios ocultos é intervir entre a entrada externa e a saída da rede de uma maneira útil. Adicionando-se uma ou mais camadas ocultas, tornamos a rede capaz de extrair estatísticas de ordem elevada.

A Figura 8 mostra um exemplo de uma RNA de 2 camadas com 4 entradas e 2 saídas.

Figura 8 – Redes alimentadas diretamente com múltiplas camadas



Fonte: adaptada de (BISHOP, 1995)

2.8.3 Algoritmo de aprendizagem por Retro propagação - *Backpropagation*

O algoritmo *Backpropagation* procura achar iterativamente a mínima diferença entre as saídas desejadas e as saídas obtidas pela rede neural, com o mínimo de erro. Dessa forma, ajustando os pesos entre as camadas através da retro propagação do erro encontrado em cada iteração (RUMELHART; HINTON; WILLIAMS, 1986).

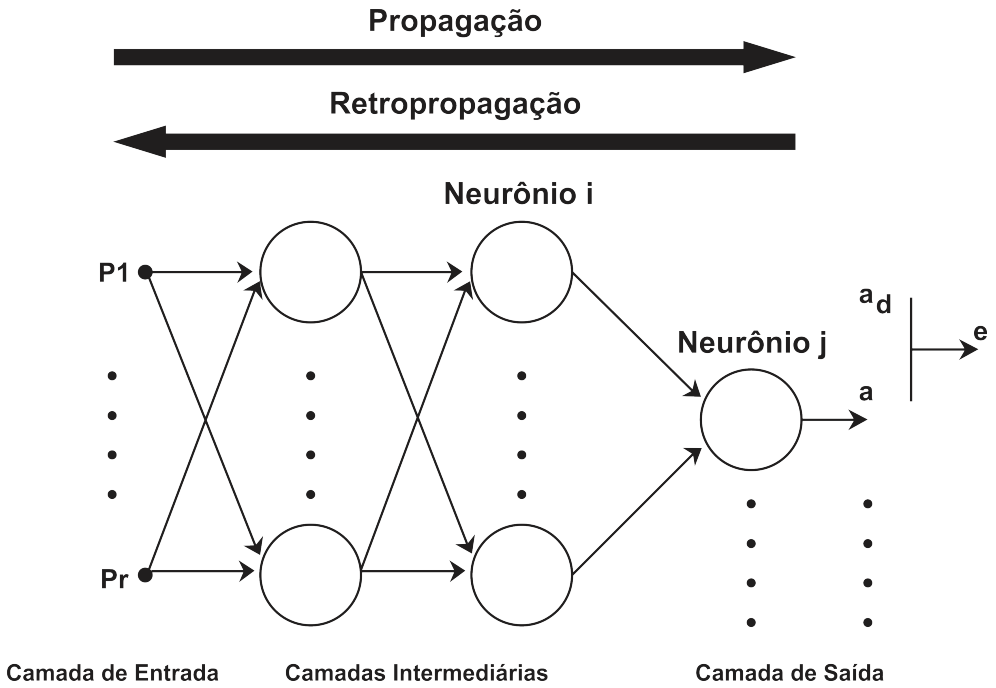
Essa regra é um dos tipos de treinamento supervisionado, em que a rede é analisada em dois casos: na sua propagação (camada por camada) e principalmente, na sua retro propagação (análise contrária à propagação), *Backpropagation*. No primeiro, os pesos sinápticos da rede são todos fixos. No segundo, os pesos são todos ajustados.

Um padrão de entrada é aplicado como um estímulo aos elementos da primeira camada da rede, que é propagado por cada uma das outras camadas até que a saída seja gerada. A saída gerada (a) é então comparada com a saída desejada (a_d), emitindo um sinal de erro (e) para cada elemento de saída. O sinal de erro é então retro propagado da camada de saída para cada elemento da camada intermediária anterior que contribui diretamente para a geração da saída.

Entretanto, cada elemento da camada intermediária recebe apenas uma porção do sinal de erro total, proporcional apenas à contribuição relativa de cada elemento na formação da saída original. Este processo se repete, camada por camada, até que cada elemento da rede receba um sinal de erro que descreva sua contribuição relativa para o erro total.

Com base no sinal de erro recebido, os pesos sinápticos são então atualizados de acordo com uma regra de correção de erro para cada elemento de modo a fazer a rede convergir para o valor de saída desejada. A Figura 9 mostra o algoritmo *Backpropagation*.

Figura 9 – Ilustração do algoritmo *Backpropagation*



Fonte: adaptada de (BISHOP, 1995)

Esse tipo de algoritmo é uma ferramenta interessante para aplicações de classificação de padrão (RAUBER, 2005). Aplicações como o objeto de estudo dessa pesquisa, as regras do controlador.

2.9 Trabalhos Relacionados

As Redes Definidas por Software estão repaginando o modelo de utilização e gerência das redes de computadores. Segundo Feamster, Rexford e Zegura (2014), o paradigma vem ganhando a atenção nos últimos anos. Além de abordar a falta de programabilidade em arquiteturas de rede existentes, permitiu a inovação na rede de uma forma mais fácil e rápida, através de sua ideia básica, que é a quebra da verticalização e da ossificação no conceito de redes, ou seja, separação em planos.

Apesar de redes SDN serem muito utilizadas na indústria e citadas em inúmeras pesquisas acadêmicas, a literatura ainda é carente de material de pesquisa que unem SDN e inteligência computacional, especialmente na tomada de decisão de regras que atuam no controlador. Trabalhos como o de (MESTRES et al., 2017) e o de (SABBEH et al., 2016), perfilam o conceito de união de SDN com elementos de inteligência computacional, porém nenhum deles trabalha com a classificação de regras do controlador SDN, que é o foco deste trabalho.

Outra pesquisa relacionada foi a de (QAZI et al., 2013) que utiliza técnicas de classificação de tráfego baseadas no aprendizado de máquina (ML) como alternativa à inspeção de pacotes. A abordagem baseada em ML não requer inspeção de carga útil de pacotes, requer um conjunto específico de recursos de nível de fluxo, como os tamanhos dos primeiros pacotes, origem e destino, portas e endereços IP. Isso geralmente resulta em um custo computacional muito menor do que o baseado em inspeção de pacotes tradicional. Não obstante, um obstáculo ao uso de ML, seria a detecção precisa e detalhada dos recursos de fluxo necessários para treinar o classificador.

Assim, o diferencial deste trabalho é a abordagem de classificação de padrões de uma rede SDN com tomada de decisão automática. Os resultados mostram que a pesquisa pode evoluir para um outro aspecto na tomada de decisão de regras, como a escolha por regras estáticas dentro do contexto mais individualizado de tratamento das mesmas, se relacionando em alguns aspectos com o trabalho de (COSTA et al., 2016), no qual testes com vários tipos de conjuntos de *match*, da tabela de fluxos, foram realizados para medir o desempenho de alguns modelos de switch. Porém neste trabalho a ideia é medir o desempenho das regras, identificando qual regra tem a melhor performance para cada caso e posteriormente classificá-las com o auxílio da inteligência computacional.

A seguir é apresentada a metodologia proposta para o desenvolvimento deste trabalho.

3 METODOLOGIA

A modelagem deste trabalho, de classificação de regras para controladores SDN, consiste na montagem de alguns cenários em laboratório para executar experimentos com Redes de Computadores, utilizando o protocolo OpenFlow e, consequentemente, SDN como base. Foram elaborados alguns cenários de avaliação com componentes de encaminhamento (através de virtualizador de rede e de um hardware-switch), algumas topologias para a SDN e modelos da RNA serão mostrados no decorrer desta seção.

A execução de experimentos com SDN gera amostras de tráfego e resultados perfilados através de regras suportadas pelo OpenFlow. Parte das amostras tornaram-se uma base de dados que alimenta uma Rede Neural Artificial – *Perceptron* Multicamadas, tal RNA foi configurada para classificar padrões e treinada para apontar qual conjunto de regras é mais eficiente para ser aplicado pelo controlador. Posteriormente, a referida base de dados é tratada e analisada com base em métricas de desempenho como atraso e quantidade de pacotes, que também serão detalhadas no decorrer da seção.

Para validar a pesquisa foram modelados, através do virtualizador de rede Mininet e em apenas uma topologia através de um hardware-switch, ambientes SDN com seus dispositivos de encaminhamento podendo operar com até 8 tipos de conjuntos de regras (de *layer* 2) do controlador POX detalhadas na subseção 3.5, sob os contextos de tráfego simulado. Foram montadas algumas topologias SDN que serão detalhadas durante a seção, as mesmas foram divididas em quatro tipos. A variação de topologias abrangendo redes definidas por software, juntamente com a variação de protocolos de diferentes *layers* serviram para testar os conjuntos de regras citados na subseção 3.5. Por se tratar de classificação de regras dinâmicas de mesmo nível ou *layer*, a variação de topologia, aplicação e quantidade de pacotes contribui para o fornecimento de uma base de dados mais consistente. E, por fim, a RNA foi testada com dois tipos de configuração, porém apenas um deles foi utilizado para o seu treinamento.

3.1 Componentes importantes no contexto de SDN

Uma visão macro da rede SDN pode ser descrita do nível mais alto para o nível mais baixo (*TOP - DOWN*). Iniciando pelas aplicações, que através da *Northbound Interface*, fornecem estatísticas processadas, eventos e requisitos para a rede, todas coletadas das abstrações da rede (topologia, etc.). Isso é possível devido a presença do controlador que é um sistema opera-

cional de rede. Tal sistema operacional controla também, através da *Southbound Interface*, os eventos, estatísticas, pacotes e comandos oriundos dos dispositivos de encaminhamento.

Dentro de redes SDN os equipamentos de controle e encaminhamento podem ser físicos e ou lógicos. Por exemplo, é possível ter um hardware-switch operando em um *rack* dentro de uma sala de telecomunicações, também é possível trabalhar com um software-switch instalado num computador pessoal e, ainda, um switch rodando em uma máquina virtual ou emulado dentro de um virtualizador de rede. No caso do controlador, existe a opção de instalá-lo no computador ou emulá-lo através de um virtualizador de rede. Nesta seção, serão abordados quais componentes de controle e encaminhamento, tanto físicos quanto lógicos serão utilizados.

3.1.1 Controlador

Para instalar as regras OpenFlow de avaliação, foi utilizado o controlador POX por se tratar do controlador mais utilizado no ambiente acadêmico na atualidade (POX, 2018). O controlador instala regras com *hard timeout* ilimitado, no momento da inicialização da rede. As regras ficam em atividade até o momento em que o switch é desconectado ou que seja alterado pelo controlador. Nos testes com switches operando em modo *legacy*, o mesmo não realiza nenhuma operação. Pois operando em modo *legacy* o equipamento se torna um switch de *layer* 2 convencional, com a função legada de criar um mapeamento relacionando endereços MAC, interfaces e portas, criando a conhecida tabela MAC.

3.1.2 Switches

Com o surgimento do OpenFlow e sua grande aceitação no mercado, várias empresas passaram a produzir equipamentos com suporte a OpenFlow, seja através de prototipação, de atualização de *firmware* ou do desenvolvimento direto dessa tecnologia. Nessa pesquisa foram avaliadas e comparadas algumas dessas implementações OpenFlow para hardware switches comerciais e software switches conhecidos, considerando a versão 1.0 do OpenFlow, com apenas uma tabela de fluxos. O hardware switch utilizado foi o *Extreme Summit x440* e o software switch utilizado foi o Open vSwitch (OvS) (PFAFF et al., 2009), ambos com suas características sumarizadas na Tabela 1.

Tabela 1 – Switches avaliados

Marca	**Modelo**	**S.O. Firmware**	**HW / SW Switch**	**Nº de Portas**	**CPU**
Extreme	Summit x440-48p	ExtremeXOS 16.2.3.5	HW	52	Single Core CPU 500 MHz
OpenvSwitch	OvS 2.7.0	Linux Debian 8.3.0	SW	-	Intel Core i5 CPU 3.1 GHz

3.1.3 Ambiente de rede virtualizado

Para aumentar as possibilidades de cenários válidos sem a utilização de laboratório físico e também pelo fato de uma rede emulada ser mais adaptável a mudanças repentinas, foi inserido um virtualizador de redes.

Para virtualizar a rede foi utilizado o Mininet, um emulador de rede com suporte a OpenFlow, capaz de criar hosts, switches, controladores e links (MININET, 2018).

Com o Mininet é possível criar uma rede e adicionar uma grande quantidade de nós em um só computador, como também é possível conectar com outras redes de outros computadores. Outro fator importante na escolha do Mininet é disponibilidade de uma Máquina Virtual - VM já configurada com o OpenFlow, POX e *Wireshark*.

3.2 Topologia da SDN

As topologias foram divididas em quatro tipos de leiautes para aumentar as possibilidades de atuação das regras dinâmicas sobre o controlador SDN.

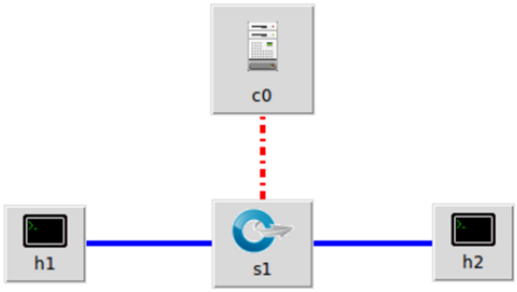
3.2.1 Topologia 1 - Composta por um controlador e um switch

Esse primeiro modelo foi considerado como padrão para os experimentos, por se tratar de uma topologia básica SDN. Com este tipo de leiaute é possível medir e comparar os resultados, variando os cenários de forma direta. Outro ponto é que esse modelo de topologia é de fácil implementação e pode ser utilizado para testar diferentes tipos de hardware-switches OpenFlow.

Desse modo, foi montada uma topologia típica no virtualizador Mininet, como mostra a Figura 10, composta por duas máquinas interligadas através de um switch OpenFlow. Uma das máquinas atua como cliente e a outra como servidor. O controlador utilizado foi o POX atuando com regras nativas citadas no decorrer da seção e o switch openflow foi o Open vSwitch.

Exclusivamente para a topologia 1, foi montada uma estrutura com um hardware switch citado na subseção 3.1 e três máquinas físicas com funções de controlador, cliente e servidor. A configuração dos computadores utilizada foi a seguinte: processador Intel Core i7-860 2.80 GHz, com 4GB de memória RAM para a máquina do cliente e processadores Intel Core i5-2400 3.1 GHz, com 8GB de memória RAM para as máquinas do controlador e do Servidor. Todos configurados com o sistema operacional Linux Debian 8.3.0.

Figura 10 – Topologia 1 - Simples - Composta por um controlador e um switch

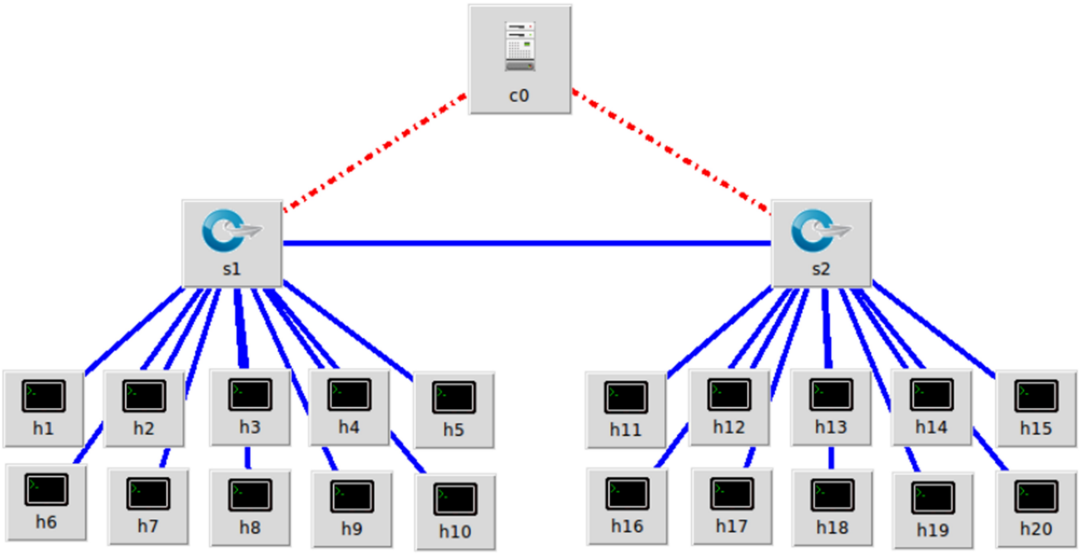


Fonte: (PRÓPRIO AUTOR)

3.2.2 Topologia 2 - Composta por um controlador e dois switches

Foi montada uma segunda topologia composta por um controlador, dois switches e vinte computadores neles interligados, formando uma rede local, como mostra a Figura 11. Todas as vinte máquinas podem atuar tanto como clientes como servidores, dependendo do enlace. Já o controlador exerce sua função de gerenciamento de regras de forma centralizada.

Figura 11 – Topologia 2 - Composta por um controlador e dois switches

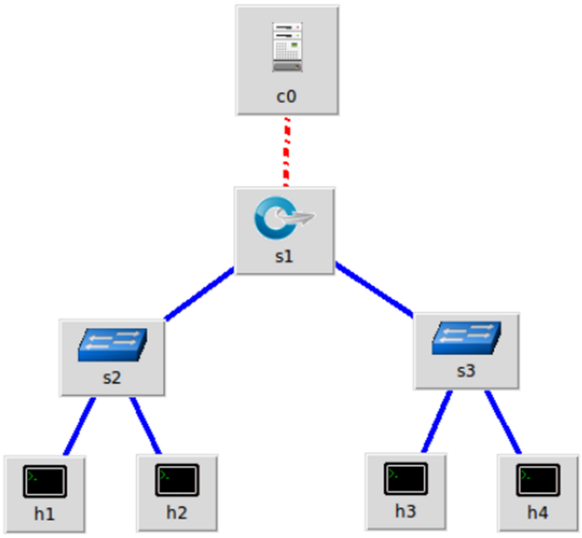


Fonte: (PRÓPRIO AUTOR)

3.2.3 Topologia 3 - Árvore de Switches

A topologia padrão, demonstrada na Figura 12, foi composta por mais de um switch OpenFlow, porém apenas um deles (s_1) possui um controlador atuando diretamente, os outros dois (s_2 e s_3) estão configurados no modo *legacy*. Essa topologia retrata uma árvore de switches ligados a dois hosts cada. Nela é possível realizar testes entre dois hosts de um mesmo switch e entre dois hosts de switches diferentes, com os dados percorrendo a árvore.

Figura 12 – Topologia 3 - Árvore de Switches

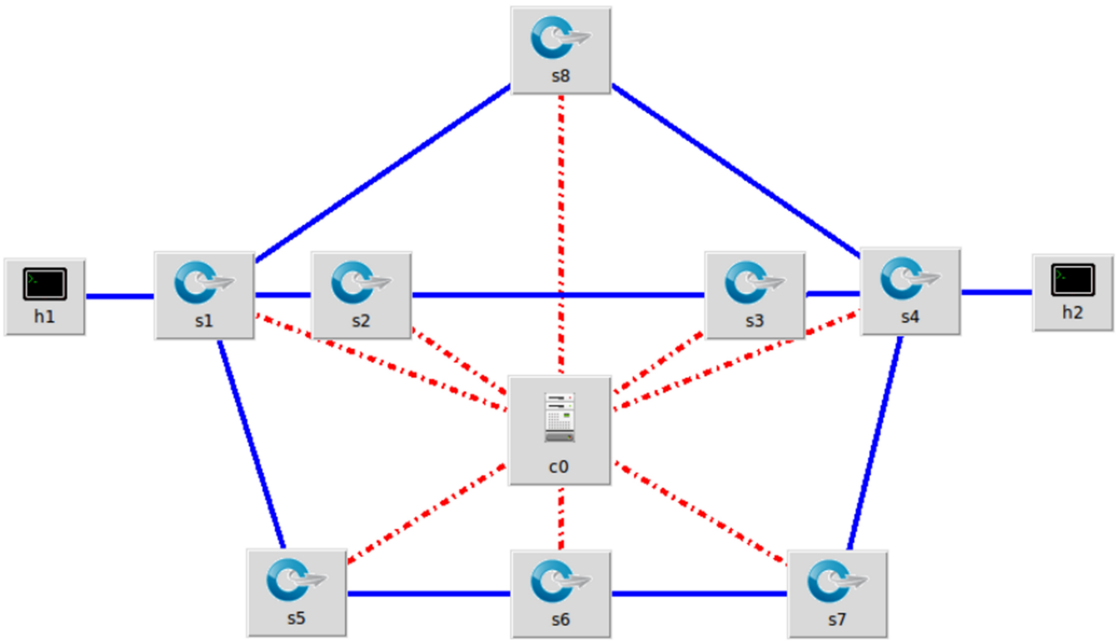


Fonte: (PRÓPRIO AUTOR)

3.2.4 Topologia 4 - Multicamadas

Por fim, na Figura 13 está representada uma topologia customizada, com intuito de demonstrar o comportamento da rede SDN sobre multicamadas, colocando em teste a performance do conjunto de regras *l2-multi* que é apropriado para este tipo de leiaute. Todos os switches estão interligados ao controlador (c_0) formando uma topologia em diamante e facilitando testes de encaminhamento de pacotes entre switch-switch e switch-controlador.

Figura 13 – Topologia 4 - Multicamadas



Fonte: (PRÓPRIO AUTOR)

3.3 Regras

Com o objetivo de realizar uma experimentação direcionada às funcionalidades de encaminhamento de pacotes, ou seja de camada 2, foram escolhidas regras para tal fim. Acresce que, outro quesito de escolha foi a compatibilidade da regra para com o tipo de controlador utilizado na pesquisa. As regras selecionadas possuem a funcionalidade de um switch de aprendizado convencional, porém com particularidades distintas que variam de acordo com o perfil da rede.

Conforme mencionado, as regras foram configuradas com o intuito de cobrir aspectos de encaminhamento. Dessa forma, foram estipuladas 3 regras de *layer* 2 preestabelecidas no controlador POX. A seguir são definidas as 3 regras, suas principais características e suas particularidades:

- *Forwarding.l2-learning* - o conjunto de regras faz com que os switches OpenFlow funcionem como switches de aprendizagem de camada dois. No entanto, ela instala fluxos extras para conexões com a mesma origem e/ou o mesmo destino. Por exemplo, conexões com origens distintas e destinos iguais possuem dois fluxos distintos.

- *Forwarding.l2-pairs* - executa de forma parecida com *Forwarding.l2-learning*, no entanto este componente é mais simples e em determinados casos mais eficiente, ele instala regras

baseadas apenas em endereços MAC, tratando a origem e o destino como um par de endereços l2.

- *Forwarding.l2-multi* - funciona como os outros switches de aprendizagem, a diferença está em ambientes com múltiplos switches. Neste caso, quando um switch aprende um endereço MAC todos os outros da rede também aprendem. É necessário a utilização de outro componente o *openflow.discovery*, que descobre qual é a topologia da rede por meio de mensagens de controle entre os switches OpenFlow, ou seja, descobre a conectividade entre os switches enviando mensagens LLDP (*Link Layer Discovery Protocol*).

Todas as 3 regras supracitadas são do tipo dinâmicas, regras dinâmicas são aquelas que podem ser instaladas pelo controlador sem a necessidade de compatibilidade de um fluxo SDN, isto é, um fluxo com campos pertencentes a determinado subconjunto. Eliminando assim o casamento conhecido como *match*, ou a compatibilidade de fluxo como dito anteriormente. As regras estáticas por sua vez são combinações de um ou mais campos de cabeçalho desenvolvidos, pelo administrador da rede, com intuito de cumprir uma ação específica, seja encaminhamento de pacotes ou até mesmo roteamento, por exemplo. Ambas podem trabalhar em conjunto.

Associado às regras dinâmicas descritas anteriormente, duas regras estáticas com funções de camada 2, também foram utilizadas. Tais regras são para dois subconjuntos de *match* individuais dentre os 12 atributos da versão 1.0 do Openflow, o subconjunto de *match* por porta do switch e o subconjunto de *match* por MAC. As regras funcionam da seguinte maneira: quando o primeiro pacote de uma solicitação chega ao switch, ele é repassado ao controlador. O controlador, inicialmente, aponta para o servidor que irá atender a requisição (normalmente o destino dos enlaces). Em seguida, o controlador cria uma entrada (ou regra) na tabela de fluxos do switch associado a esse pacote, que define a ação de encaminhamento do pacote para a saída específica associada ao servidor. Vale ressaltar que os demais pacotes pertencentes à mesma solicitação, quando chegarem ao switch, já encontrarão as regras de encaminhamento instaladas previamente, além do apoio das regras dinâmicas supracitadas, não sendo necessário o repasse dos mesmos ao controlador.

Todas as regras supracitadas são de controladores nativos existentes no POX, foram desenvolvidas para fins didáticos e refletem diferentes cenários de encaminhamento de pacotes relacionados a variáveis que compõem a tabela de fluxo como: portas do switch, endereços MAC e IP, VLAN, ToS, portas TCP e outros. E conforme mencionado anteriormente, nesse trabalho, a utilização de cabeçalhos da tabela de fluxos se restringiu a portas do switch e ende-

reços MAC, através da utilização de regras estáticas associadas à subconjuntos de *match* para tais fins.

Outra funcionalidade utilizada junto ao conjunto de regras dinâmicas é a opção de acrescentar componentes, tal como, o *open-flow-spanning-tree* que cria uma *Spanning Tree* da rede que será executada caso a regra escolhida não suporte rede com loops (NOX, 2018).

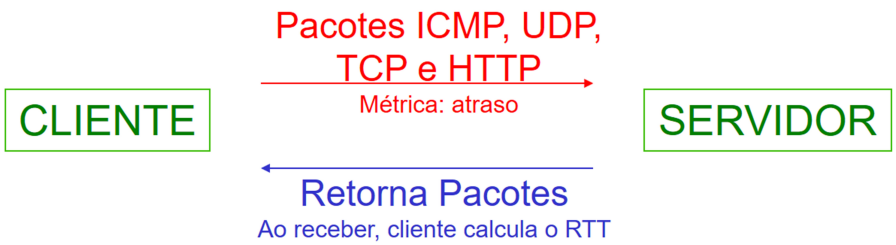
3.4 Carga e Métricas

Para os testes foram enviados pacotes, gerados em uma SDN com regras de *layer* 2 do controlador POX, entre a máquina cliente e o servidor através do switch OpenFlow a ser avaliado. A métrica de atraso foi medida variando quantidade de pacotes, protocolo e topologia. O atraso foi coletado com base no retorno do servidor para o cliente que calcula o *Round Trip Time* (RTT) de cada um dos pacotes.

Para geração da carga de pacotes foi utilizado o gerador de tráfego BWPING (ALVES et al., 2018), no inicio dos testes, exclusivamente com tráfego UDP. E, posteriormente, foi utilizado o gerador OSTINATO (OSTINATO, 2018), para geração de tráfego ICMP, TCP, HTTP. Depois de alguns experimentos e pelo fato de que o BWPING só gera tráfego UDP, optou-se em padronizar e utilizar apenas o OSTINATO para geração de tráfego com todos os protocolos (ICMP, UDP, TCP e HTTP). Todo o tráfego gerado passa obrigatoriamente por pelo menos um switch OpenFlow. Estabeleceu-se um padrão de tamanho dos pacotes de 128 bytes. Também foi estabelecido alguns tamanhos da janela de envio de pacotes, ou seja, a quantidade de pacotes que serão enviados em rajada, divididos em pequeno, médio e grande, respectivamente: 100, 1.000 e 10.000 pacotes. Além da variação das 4 topologias citadas na subseção 3.2.

As aplicações de teste foram desenvolvidas conforme mostrado na Figura 14, na qual a coleta dos atrasos foi feita variando os protocolos ICMP, UDP, TCP e HTTP.

Figura 14 – Aplicação de teste – Cliente e Servidor



Fonte: (PRÓPRIO AUTOR)

3.4.1 Operação de *FlowStats*

O *FlowStats* é uma operação de apoio e diagnóstico de tráfego que fornece estatísticas de fluxo em uma rede SDN. Ela é oriunda de um código construído em *Python*, nativo do controlador POX (NOX, 2018), que se encontra disponível em seu repositório na internet. Tal código é editável e pode ser adaptado às necessidades de pesquisadores e desenvolvedores. Nesse trabalho, modificações foram inseridas para atender os requisitos das regras dinâmicas e estáticas que foram utilizadas nos experimentos que serão detalhados na subseção 3.6.

Dentre as opções de estatística de fluxo está a coleta da métrica de atraso. Nos testes para composição dos traces mencionados anteriormente, o controlador requisita estatísticas (*FlowStats*) ao switch uma vez a cada segundo (ou a cada pacote enviado, dependendo da configuração), e calcula, posteriormente, o atraso de cada resposta enviada pelo switch. Vale ressaltar que esses testes foram executados com a variação da carga de trabalho a qual o switch foi submetido (carga baixa, média ou alta - 100, 1.000 ou 10.000 pacotes) e a variação de protocolos (ICMP, UDP, TCP e HTTP) e de tipos diferentes de topologia (topologias 1, 2, 3 e 4). Observar o desempenho dos switches nas operações de *FlowStats* foi importante para definir se esse é o método mais vantajoso para obtenção de dados estatísticos em um esquema com variações no switch (por exemplo, variação de carga), ou se o método mais vantajoso é requisitar estatísticas de rede de cada servidor (COSTA et al., 2016).

Todos os testes referentes aos cenários supracitados foram fundados no envio alternado de 100, 1.000 e 10.000 pacotes ICMP, UDP, TCP e HTTP do cliente para o servidor através do switch OpenFlow avaliado. A taxa de envio é de 10 pacotes por segundo. Os resultados obtidos são compostos pelos valores dos atrasos individuais de cada pacote e, caso necessário, existe a possibilidade de calcular os *jitters* correspondentes, além do cálculo das médias e desvios padrões dos conjuntos de atrasos, e os cálculos dos intervalos de confiança desses conjuntos. Neste trabalho, foram utilizados intervalos de confiança de 95% para uma distribuição normal de amostras.

3.5 Aplicações

Os protocolos escolhidos foram o ICMP, UDP, TCP e HTTP, pois refletem diferentes aspectos de aplicações e atuam em diferentes níveis de camadas. Como por exemplo, enlace, rede, transporte e aplicação. Apesar do tráfego gerado ser baseado nesses protocolos/aplica-

ções, todas as regras atuantes no controlador são de camada 2 e sua classificação inteligente é feita com base na performance de cada regra sob esse aspecto. A medição de performance de regras dinâmicas necessita de protocolos que atuem em diferentes camadas, desde o enlace até a aplicação (KREUTZ et al., 2015).

3.6 Cenários de avaliação das Regras

O modelo de avaliação do desempenho das regras é base para todo o treinamento e criação do mecanismo inteligente de classificação das mesmas. A metodologia para o desenvolvimento de tal modelo foi fundamentada nos cenários apresentados na Tabela 2 que foram inseridos na elaboração de um algoritmo de referência para as rotinas dos experimentos.

Tabela 2 – Metodologia de testes

Cenário	Descritivo
1	Combinar os três conjuntos de regras dinâmicas de *layer 2* com dois tipos de regras estáticas, também de *layer 2*
2	Combinar os diferentes tipos de topologia, protocolo e quantidade de pacotes com as regras
3	Adaptar a operação de *flowstats* para captura da métrica de atraso
4	Realizar o pós-processamento dos atrasos gerados
5	Transformar os protocolos, as topologias e a quantidade de pacotes em valores numéricos fixos
6	Coletar os valores de atraso, topologia, protocolo e quantidade de pacotes para formação da tabela de entrada da RNA
7	Configurar e treinar a RNA
8	Realizar os mesmos testes com hardware-switch
9	Comparar performance de um hardware-switch para com o softwares-switch
10	Comparar performance do método tradicional de inserção de regras com o gerado pela RNA
11	Medir o custo operacional gerado com a troca de regras no controlador

No cenário 1, foi feito uma combinação entre as 3 regras dinâmicas (*learning*, *pairs* e *multi*) com um regra estática para subconjunto de *match* por porta, ou seja, a *learning* foi setada juntamente com a regra estática de porta, em seguida a *pairs*, também, com a regra estática de porta e por fim a *multi* com a mesma regra estática, as 3 etapas feitas separadamente. O mesmo procedimento foi realizado para a regra estática com subconjunto de *match* por MAC.

O cenário 2 foi realizado com base no primeiro, ou seja, para cada uma das duas etapas do cenário 1 uma combinação de topologia, protocolo e quantidade de pacotes foi aplicada. Na prática, foi colocado em funcionamento uma regra dinâmica (por exemplo, a *learning*) junto com uma regra estática (por exemplo, a de *match* por porta), disparando uma carga de 100 pacotes ICMP utilizando a topologia 1. Equitativamente, foram realizadas todas as combinações possíveis dentre as variações propostas.

Para avaliação de desempenho das regras de encaminhamento de *layer* 2 no controlador, no cenário 3, os valores de atraso oriundos da operação de *flowstats*, foram colocados como uma estatística exclusiva dentro do código, para facilitar a coleta posterior.

Com a realização dos experimentos contidos nos cenários 1 e 2 e com a coleta dos atrasos conforme mencionado no cenário 3, tornou-se necessário o pós-processamento da base de dados gerada. Isso aconteceu para adequá-la aos moldes de entrada do sistema inteligente de classificação de regras.

Na sequência, também, foi necessário realizar uma conversão dos valores fixos de topologia (topologias 1, 2, 3 e 4), protocolo (ICMP, UDP,TCP e HTTP) e quantidade de pacotes (100, 1.000 e 10.000 pacotes) em valores numéricos igualmente fixos. Tal processo se deu devido à uma necessidade de adaptação da tabela de entrada da RNA. Os valores foram convertidos como demonstrado a seguir:

- Topologia 1 -> Valor = 0,1
- Topologia 2 -> Valor = 0,2
- Topologia 3 -> Valor = 0,3
- Topologia 4 -> Valor = 0,4
- Protocolo ICMP -> Valor = 0,5
- Protocolo UDP -> Valor = 0,6
- Protocolo TCP -> Valor = 0,7
- Protocolo HTTP -> Valor = 0,8
- Quantidade de pacotes em 100 -> Valor = 0,90
- Quantidade de pacotes em 1.000 -> Valor = 0,95
- Quantidade de pacotes em 10.000 -> Valor = 0,99

Na configuração dos experimentos desse trabalho, os valores de entrada da RNA são obrigatoriamente valores numéricos entre 0 e 1. Tais *inputs*, quando utilizados como entradas de classificação de padrões, necessitam ser entradas que assumam valores fixos (MASTERS, 1993). Os números de 0,1 até 0,99 foram escolhidos para se distinguirem visualmente dos demais (que são números muito pequenos, com várias casas decimais depois vírgula), quando forem utilizados dentro das tabelas no processo de treinamento e validação da rede neural artificial.

No cenário 6 foi feita a junção dos valores de formação de uma tabela que serviu de entrada da RNA. A tabela possui 4 colunas: atraso, topologia, protocolo e quantidade de pacotes e seus valores são números reais entre 0 e 1. Para uma RNA ativa, porém não atuando diretamente em uma rede em produção, os valores foram coletados separadamente e depois foram juntados todos na referida tabela, para a preparação de cada um dos treinamentos realizados.

No cenário 7, foram executados a configuração e o treinamento da RNA. Etapas que serão abordadas na seção 4.

No cenário 8 foram realizados testes com o switch da *Exteme*, apresentado na Tabela 1, com intuito de avaliar o desempenho de um hardware-switch, compará-lo ao software-switch e também para suprir a necessidade de montagem de um cenário para os primeiros testes com uma classificação inteligente de regras a quente, ou seja, em uma rede em produção. Todos os testes realizados foram feitos da mesma maneira dos feitos com o virtualizador Mininet, todavia exclusivamente para o topologia 1. No cenário 9, os resultados dos experimentos com o hardware-switch e com o software-switch foram comparados e serão mostrados na próxima seção.

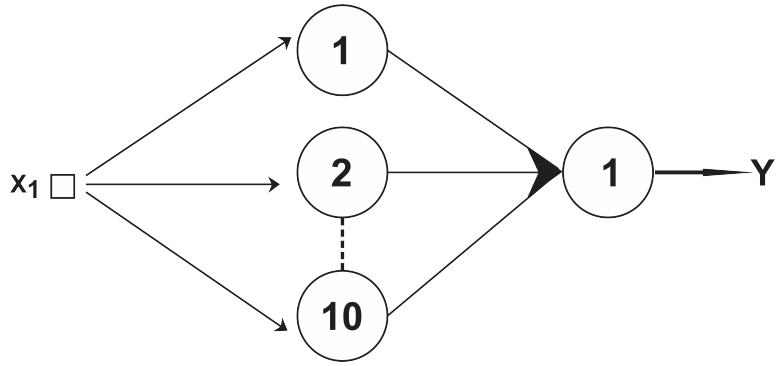
Os dois últimos cenários, 10 e 11, tiveram como objetivo comparar o método tradicional de inserção de regras com os resultados gerados pela RNA e medir o custo operacional gerado com a troca de regras no controlador. Ainda que, nesse trabalho, a troca de regras é feita sempre pelo administrador da rede, tendo apenas como sugestão a resposta do sistema inteligente.

3.7 Configurações dos experimentos com a RNA

A Rede Neural Artificial, neste trabalho, é uma ferramenta de classificação de padrões que utiliza o algoritmo de aprendizagem *backpropagation*, com amostras de treinamento apoiadas no comportamento da rede SDN.

Por questões operacionais foi utilizado, inicialmente, uma rede perceptron com uma entrada e uma saída para a adaptação do comportamento entre fluxos monitorados e as regras aplicadas pelo controlador, conforme apresentada na Figura 15. Nessa topologia são dez neurônios na camada oculta.

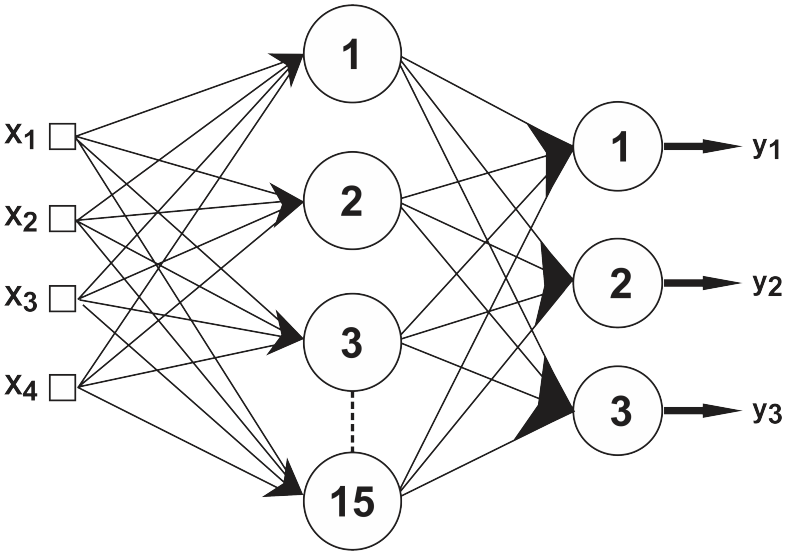
Figura 15 – Topologia MLP – uma entrada e uma saída



Fonte: (PRÓPRIO AUTOR)

Em seguida foi utilização outra topologia com quatro entradas e três saídas conforme Figura 16, trazendo uma melhoria na capacidade da RNA, podendo nessa configuração classificar três tipos de conjuntos de regras para o controlador sistematizadas nas entradas demonstradas na subseção a seguir. Nesta topologia são quinze neurônios na camada oculta.

Figura 16 – Topologia MLP – quatro entradas e três saídas



Fonte: (PRÓPRIO AUTOR)

3.7.1 Testes para a tomada de decisão automática com a RNA

A motivação de testes com o modelo de inteligência computacional em uma rede em funcionamento é devido à possibilidade de conseguir resultados significativos e que pudessem confirmar a utilização dos mesmos no desenvolvimento de um protótipo de classificação automática de regras em uma rede em produção. Testes a quente proporcionam reconfiguração da

rede e experimentação, retornando valores que nos permitem optar pelo que for mais vantajoso na operação e deixando o fluxo de comunicação fim-a-fim mais rápido, por exemplo.

Os testes a quente neste trabalho ficaram restritos a elaboração de uma rotina de experimentação para tornar viável uma implementação e desenvolvimento futuro. Para tal, os testes ficaram limitados a um tipo de topologia, a topologia 1 que é a mais simples de ser implementada com equipamentos físicos e que possui apenas um switch OpenFlow. Os valores de entrada da RNA foram coletados dos experimentos com o hardware-switch.

Algumas adaptações tiveram que ser realizadas nas etapas listadas na subseção 3.6, a seguir serão detalhados os procedimentos para construção do algoritmo de execução do mecanismo que automatizará a rotina de alimentação e coleta de resultados da RNA, toda execução precisará ser feita após a configuração e treinamento da RNA.

1. Adaptar a operação de *flowstats* para captura da métrica de atraso, valores de protocolos e valores de quantidade de pacotes todos em um único arquivo no formato de texto. Tal adaptação foi realizada através da alteração do código padrão *flowstats.py*, construído em *Python* e disponibilizado no repositório do POX (POX, 2018), editando o código para atender os requisitos de entrada da RNA.

2. Transformar os protocolos, a quantidade de pacotes em valores numéricos: os valores 0,5; 0,6; 0,7 e 0,8 para protocolo ICMP, UDP, TCP e HTTP respectivamente. E os valores 0,90; 0,95 e 0,99 para quantidade de pacotes em 100, 1.000 e 10.000 respectivamente. Adotar um critério de conversão de valores de acordo com a carga de pacotes da rede, ou seja, se a carga de pacotes for baixa (valores entre 0 e 100 pacotes) converter para o valor numérico 0,90; se a carga de pacotes for média (valores entre 101 e 1.000 pacotes) converter para o valor numérico 0,95; e se a carga de pacotes for alta (valores entre 1.001 e 10.000 pacotes) converter para o valor numérico 0,99.

3. Inserir a topologia 1 com o valor numérico fixo em 0,1.

4. Adaptar os valores gerados em formato de tabela com as colunas topologia (fixa em 0,1), protocolo (com os valores variando em 0,5; 0,6; 0,7 e 0,8), quantidade de pacotes (com os valores variando em 0,90; 0,95 e 0,99) e atraso médio em milissegundos.

5. Combinar os passos 3 e 4 em um só arquivo de texto, formando a Tabela 4 (porém os valores de cabeçalho são excluídos do arquivo de texto).

6. Alimentar a RNA, inserindo como entrada o arquivo de texto do passo 5.

7. Enviar um *pop-up* na tela do operador de rede alertando-o para a troca de regra.

A seguir é mostrada a entrada formatada da RNA através da Tabela 3.

Tabela 3 – Valores numéricos de entrada da RNA

Qtde Pkt	**Atraso**	**Topologia**	**Protocolo**
x1	**x2**	**x3**	**x4**
0,99	0,280460	0,1	0,1
0,90	0,004096	0,1	0,5
0,90	0,151722	0,1	0,7
0,90	0,006379	0,1	0,5
0,95	0,350794	0,1	0,7
0,90	0,009071	0,1	0,7
0,90	0,035979	0,1	0,6
0,90	0,001881	0,1	0,5
0,95	0,032780	0,1	0,7
0,90	0,053361	0,1	0,6
0,90	0,006790	0,1	0,5
0,90	0,007084	0,1	0,5
0,90	0,318858	0,1	0,8
0,90	0,065447	0,1	0,6
0,90	0,035074	0,1	0,6
0,99	0,211026	0,1	0,5
0,90	0,012675	0,1	0,6
0,99	0,341447	0,1	0,6

Os valores listados na Tabela 3 são um exemplo de como é uma tabela de entrada da rede neural artificial, isto é, x1, x3 e x4 são valores numéricos fixos entre 0 e 1 representando a carga de pacotes, a topologia e o protocolo, respectivamente. E a entrada x2 são exemplos de valores de atraso em milissegundos.

Com base nas 7 etapas anteriores foram desenvolvidos dois pseudocódigos, para serem implementado na linguagem *python*. A linguagem *python* foi escolhida por ser a mesma utilizada no desenvolvimento do controlador POX, inclusive no código da operação de *flowstats* (utilizado nos experimentos). A seguir estão descritos o Algoritmo 1 para coleta de estatísticas e o Algoritmo 2 para sugestão das regras ao operador.

Algoritmo 1 COLETA DE ESTATÍSTICA DE FLUXO

Entrada: $PerfilEntrada, Evento$
Saída: $QtdPacotes, Atraso, Topologia, Protocolo$
início
    $pacotes = 0$
    for f : $Evento.estatistca$ **do**
        $pacotes$ += f.contadorPacotes
    end
    if $pacotes \leq 100$ **then**
        $QtdPacotes = 0.9$
    end
    else if $pacotes \leq 1000$ **then**
        $QtdPacotes = 0.95$
    end
    else
        $QtdPacotes = 0.99$
    end
    $TempoInicial$ = tempoAtual()
    $Atraso$ = tempoAtual() $-TempoIncial$

    if $Evento.Pacote = ICMP$ **then**
        $Protocolo = 0.5$
    end
    else if $Evento.Pacote = UDP$ **then**
        $Protocolo = 0.6$
    end
    else if $Evento.Pacote = TCP$ **then**
        $Protocolo = 0.7$
    end
    else if $Evento.Pacote = HTTP$ **then**
        $Protocolo = 0.8$
    end
    if $TopologiaEntrada = 1$ **then**
        $Topologia = 0.1$
    end
    else if $TopologiaEntrada = 2$ **then**
        $Topologia = 0.2$
    end
    else if $TopologiaEntrada = 3$ **then**
        $Topologia = 0.3$
    end
    else
        $Topologia = 0.4$
    end
fim
retorna $QtdPacotes, Atraso, Topologia, Protocolo$

Algoritmo 2 SUGESTÃO DE REGRA

```
início
    for i : infinito do
        MatrizEstatistica = LerArquivo(EstatisticaLearning) + LerArquivo(EstatisticaPairs)
        +LerArquivo(EstatisticaMulti)

        Regra = RNA(MatrizEstatistica);

        if Regra = 'Learning' then
            ExibirSugestao('Learning recomendável');
        end
        else if Regra = 'Pairs' then
            ExibirSugestao('Pairs recomendável');
        end
        else
            ExibirSugestao('Multi recomendável');
        end
    end
fim
retorna SugestaoDeRegra
```

O Algoritmo 1 executa a conversão dos valores de topologia, quantidade de pacotes e protocolo todos oriundos da operação de *flowstats*, coletando assim a estatística de fluxo já formatada nos padrões de entrada da RNA, para cada tipo de regra dinâmica avaliada. Já o Algoritmo 2 executa em *loop* a alimentação de uma matriz com valores das estatísticas de fluxos das 3 regras e compara com a saída da inteligência artificial. Se os valores forem compatíveis com a regra *l2-learning*, exibe a sugestão para o operador da referida regra. Se os valores forem compatíveis com a regra *l2-pairs*, exibe a sugestão para o operador da *l2-pairs* e se os valores forem compatíveis com a regra *l2-multi*, exibe a sugestão para o operador da *l2-multi*.

Por se tratar de testes iniciais, para evitar custos operacionais oriundos da troca de regras no controlador e para manter a autonomia do ser humano, optou-se em informar para o profissional de rede que estiver operando a possibilidade de troca como sugestão. A opção de utilizar alertas nos supervisórios de operação é prática comum para adoção de qualquer sistema artificial de inteligência computacional (STANLEY; MIIKKULAINEN, 2002).

A seguir são apresentados os resultados deste trabalho.

4 RESULTADOS

A seguir são abordados os resultados dos experimentos configurados de acordo com a metodologia apresentada. Detalhes de testes com a SDN, com a RNA e um comparativo entre o modelo de classificação de regras tradicional e o modelo sugerido pela rede neural artificial também são mostrados. Paralelamente, também será explanada a inserção de métodos de elaboração de treinamento e validação, que se ausentaram da seção anterior pelo motivo de que nessa parte do trabalho o leitor já possui o embasamento necessário para interpretar, de modo mais completo, tais configurações. Outro motivo é que os detalhes citados, sendo descritos nessa parte do texto, respeitam uma sequência cronológica de configurações da RNA.

4.1 Experimentos com a SDN

Inicialmente a proposta foi avaliar a performance do conjunto de regras oriundas do código *l2-learning.py* presente no diretório POX (*pox/pox/forwarding/l2-learning.py*), partindo do pressuposto de que o referido controlador esteja instalado, independente de ser máquina física ou virtual. Este código, faz com que o Controlador, dinamicamente, saiba as relações entre MAC e interface dos hosts conectados a ele, além de inserir as regras nos switches pelo protocolo OpenFlow. Tal código vem disponível com a instalação do controlador POX.

Outros dois conjuntos de regras nativas do controlador POX também foram avaliadas nesse cenário: os códigos *l2-pairs.py* e o *l2-multi.py* que depende do *openflow.discovery*. Além da regra estática do subconjunto de *match* por porta e da regra estática do subconjunto de *match* por MAC.

Cada um dos três conjuntos de regras supracitados foram avaliados da seguinte forma: variando as quatro topologias, os quatro protocolos (ICMP, UDP, TCP e HTTP) e medindo métricas de atraso. Por fim, variando, também, o número de pacotes em três tamanhos diferentes: pequeno, médio e grande (respectivamente 100, 1.000 e 10.000 pacotes). Todo o tráfego foi gerado com pacotes oriundos das ferramentas BWPING e OSTINATO, possibilitando assim a coleta das métricas abordadas. A variação desses parâmetros influenciam e geram uma modelagem de diferentes perfis de rede, deixando os resultados mais consistentes pois, dessa forma, se aproximam da diversidade encontrada nos ambientes reais.

4.1.1 Testes com regra estática do subconjunto de *match* por porta

Com o objetivo de aumentar o nível de complexidade, acrescentando funcionalidades ao processo de encaminhamento de pacotes, foram inseridas 2 regras estáticas no contexto. A primeira delas foi a regra estática do conjunto de *match* por porta. Na Tabela 4 estão listados os resultados dos intervalos de confiança para os atrasos em milissegundos, utilizando a quantidade de pacotes em 100, variando as topologias e os protocolos. Isso para a *l2-learning.py*, para a *l2-pairs.py* e para a *l2-multi.py* todas associadas à regra estática com conjunto de *match* por porta.

Tabela 4 – Intervalo de confiança da latência para experimentos com carga baixa de pacotes (em ms) - *match* por porta

Qtde Pkt	topo	Protocolo	learning	pairs	multi
100	1	ICMP	0,001245 - 0,001263	0,001790 - 0,001804	0,002290 - 0,002304
100	2	ICMP	0,002792 - 0,002806	0,003289 - 0,003303	0,003791 - 0,003805
100	3	ICMP	0,004296 - 0,004311	0,004796 - 0,004810	0,005291 - 0,005305
100	4	ICMP	0,005798 - 0,005813	0,006298 - 0,006313	0,006792 - 0,006807
100	1	UDP	0,009483 - 0,009519	0,015221 - 0,015539	0,022908 - 0,023051
100	2	UDP	0,027970 - 0,028114	0,032872 - 0,033016	0,037963 - 0,038103
100	3	UDP	0,047894 - 0,048036	0,052846 - 0,052986	0,042974 - 0,043120
100	4	UDP	0,062902 - 0,063043	0,067917 - 0,068064	0,057945 - 0,058090
100	1	TCP	0,042401 - 0,042586	0,047885 - 0,048028	0,052983 - 0,053127
100	2	TCP	0,062886 - 0,063028	0,057902 - 0,058047	0,067944 - 0,068086
100	3	TCP	0,077898 - 0,078041	0,072874 - 0,073015	0,082867 - 0,083014
100	4	TCP	0,097950 - 0,098091	0,092925 - 0,093070	0,087891 - 0,088033
100	1	HTTP	0,112227 - 0,113117	0,159741 - 0,160460	0,194701 - 0,195069
100	2	HTTP	0,252408 - 0,252585	0,214334 - 0,215394	0,294323 - 0,294715
100	3	HTTP	0,354774 - 0,355137	0,313384 - 0,314390	0,384866 - 0,385225
100	4	HTTP	0,494665 - 0,495014	0,464809 - 0,465166	0,407187 - 0,407731

O conjunto de regras dinâmicas com melhor desempenho variou de acordo com a topologia e com protocolo utilizado, independente da quantidade de pacotes. De forma geral, a resposta de cada experimento seguiu os moldes das funções pertinentes ao seu respectivo conjunto de regras. Por exemplo, o código *l2-learning* insere um fluxo extra quando uma dada origem e/ou destino são iguais, contudo mostrou um resultado mais lento em termos de atraso quando encontrou protocolos TCP e HTTP, perdeu performance com o protocolo UDP enquanto que com o protocolo ICMP obteve o melhor desempenho em todas as topologias.

O código *l2-pairs* segue comportamento similar com o *l2-learning*, porém foi melhor do que o *l2-learning* com a utilização dos protocolos TCP e HTTP. Já o código *l2-multi* obteve

o melhor resultado quando utilizou-se a topologia com o maior número de switches, a topologia 4. Também foi melhor na topologia 3 com o protocolo UDP.

Para a quantidade de pacotes em 1.000 foi utilizada a mesma metodologia de experimentos aplicada à quantidade de pacotes em 100. Os resultados para a quantidade pacotes em 1.000 foram os seguintes:

- **ICMP - Topologias 1, 2 e 3** - a *l2-learning.py* obteve a melhor performance, em seguida a *l2-pairs.py* e por último a *l2-multi.py*;
- **ICMP - Topologia 4** - a *l2-multi.py* obteve a melhor performance, em seguida a *l2-pairs.py* e por último a *l2-learning.py*;
- **UDP - Topologia 1** - a *l2-learning.py* obteve a melhor performance, em seguida a *l2-pairs.py* e por último a *l2-multi.py*;
- **UDP - Topologias 2 e 3** - a *l2-pairs.py* obteve a melhor performance, em seguida a *l2-learning.py* e por último a *l2-multi.py*;
- **UDP - Topologia 4** - a *l2-multi.py* obteve a melhor performance, em seguida a *l2-pairs.py* e por último a *l2-learning.py*;
- **TCP - Topologia 1** - a *l2-learning.py* obteve a melhor performance, em seguida a *l2-pairs.py* e por último a *l2-multi.py*;
- **TCP - Topologia 2** - a *l2-pairs.py* obteve a melhor performance, em seguida a *l2-learning.py* e por último a *l2-multi.py*;
- **TCP - Topologias 3 e 4** - a *l2-multi.py* obteve a melhor performance, em seguida a *l2-learning.py* e por último a *l2-pairs.py*;
- **HTTP - Topologias 1, 2, 3 e 4** - obtiveram o mesmo ranqueamento que o protocolo TCP;

A Tabela 5 lista os intervalos de confiança para os atrasos em milissegundos, utilizando a quantidade de pacotes em 1.000, variando as topologias e os protocolos. Isso para a *l2-learning.py*, para a *l2-pairs.py* e para a *l2-multi.py* todas associadas à regra estática com conjunto de *match* por porta.

Tabela 5 – Intervalo de confiança da latência para experimentos com carga média de pacotes (em ms) - *match* por porta

Qtde Pkt	Topo	Protocolo	learning	pairs	multi
1000	1	ICMP	0,003245 - 0,003256	0,003797 - 0,003806	0,004293 - 0,004302
1000	2	ICMP	0,004801 - 0,004809	0,005294 - 0,005303	0,005801 - 0,005809
1000	3	ICMP	0,006295 - 0,006304	0,006796 - 0,006805	0,007293 - 0,007301
1000	4	ICMP	0,008745 - 0,008755	0,007717 - 0,008045	0,008300 - 0,008308
1000	1	UDP	0,010976 - 0,011017	0,015487 - 0,015508	0,019480 - 0,019501
1000	2	UDP	0,026247 - 0,026257	0,022484 - 0,022504	0,029478 - 0,029498
1000	3	UDP	0,034745 - 0,034755	0,030493 - 0,030514	0,038745 - 0,038756
1000	4	UDP	0,049429 - 0,049451	0,045974 - 0,045994	0,041227 - 0,041278
1000	1	TCP	0,062434 - 0,062537	0,067975 - 0,068058	0,075434 - 0,075616
1000	2	TCP	0,087983 - 0,088065	0,082961 - 0,083043	0,095386 - 0,095573
1000	3	TCP	0,157430 - 0,157739	0,113714 - 0,114298	0,194400 - 0,194588
1000	4	TCP	0,294320 - 0,294546	0,254852 - 0,255057	0,209775 - 0,210186
1000	1	HTTP	0,307353 - 0,307664	0,357327 - 0,357634	0,392268 - 0,392579
1000	2	HTTP	0,462437 - 0,462541	0,409998 - 0,410406	0,494821 - 0,495028
1000	3	HTTP	0,619796 - 0,621846	0,594821 - 0,595027	0,524861 - 0,525068
1000	4	HTTP	0,649958 - 0,650165	0,694282 - 0,694508	0,604939 - 0,605144

A Tabela 6 lista os intervalos de confiança para os atrasos em milissegundos, utilizando a quantidade de pacotes em 10.000, variando as topologias e os protocolos. Isso para a *l2-learning.py*, para a *l2-pairs.py* e para a *l2-multi.py* também associadas à regra estática com conjunto de *match* por porta.

Tabela 6 – Intervalo de confiança da latência para experimentos com carga alta de pacotes (em ms) - *match* por porta

Qtde Pkt	topo	Protocolo	learning	pairs	multi
10000	1	ICMP	0,0109880 - 0,0110110	0,0154910 - 0,0155020	0,0194870 - 0,0194980
10000	2	ICMP	0,0224920 - 0,0225030	0,0262450 - 0,0262500	0,0294900 - 0,0295010
10000	3	ICMP	0,0387470 - 0,0387520	0,0304920 - 0,0305040	0,0347490 - 0,0347550
10000	4	ICMP	0,0494460 - 0,0494580	0,0412310 - 0,0412590	0,0459930 - 0,0460040
10000	1	UDP	0,0499800 - 0,0500020	0,0579580 - 0,0580030	0,0639880 - 0,0640100
10000	2	UDP	0,0774580 - 0,0775140	0,0679830 - 0,0680280	0,0829670 - 0,0830120
10000	3	UDP	0,0919930 - 0,0920150	0,0869970 - 0,0870200	0,0979050 - 0,0979490
10000	4	UDP	0,1944270 - 0,1945510	0,1574360 - 0,1576070	0,1098830 - 0,1101090
10000	1	TCP	0,2073400 - 0,2075100	0,2524180 - 0,2525870	0,2948780 - 0,2949900
10000	2	TCP	0,3549730 - 0,3550860	0,3049160 - 0,3050300	0,3904810 - 0,3905150
10000	3	TCP	0,4624420 - 0,4624990	0,4974280 - 0,4974840	0,4099420 - 0,4101690
10000	4	TCP	0,5549550 - 0,5550670	0,5954680 - 0,5955700	0,5174720 - 0,5175280
10000	1	HTTP	0,6073960 - 0,6075640	0,6574820 - 0,6575390	0,6948400 - 0,6949520
10000	2	HTTP	0,7549080 - 0,7550210	0,7049440 - 0,7050580	0,7954690 - 0,7955260
10000	3	HTTP	0,8624520 - 0,8625090	0,8974240 - 0,8974790	0,8099040 - 0,8101300
10000	4	HTTP	0,9549690 - 0,9550830	0,9954360 - 0,9955370	0,9174790 - 0,9175360

Os resultados do ranqueamento (ou seja, qual regra dinâmica obteve melhor desempenho em termos de atraso médio, seguido da segunda colocada e por fim a terceira colocação) para a quantidade pacotes em 10.000 foram iguais aos dos experimentos com 1.000 pacotes, com exceção da topologia 3 para o protocolo ICMP, onde a *l2-pairs.py* obteve a melhor performance, em seguida a a *l2-multi.py* e por último a *l2-learning.py*.

De forma geral, a resposta de cada experimento seguiu as funcionalidades pertinentes ao seu respectivo conjunto de regras. Por exemplo, o código *l2-learning* insere um fluxo extra quando uma dada origem e/ou destino são iguais, contudo mostrou um resultado mais lento em termos de atraso quando encontrou uma carga maior de pacotes, porém, foi o código que obteve melhor desempenho quando encontrou carga com quantidade de pacotes em 100 e com o protocolo ICMP.

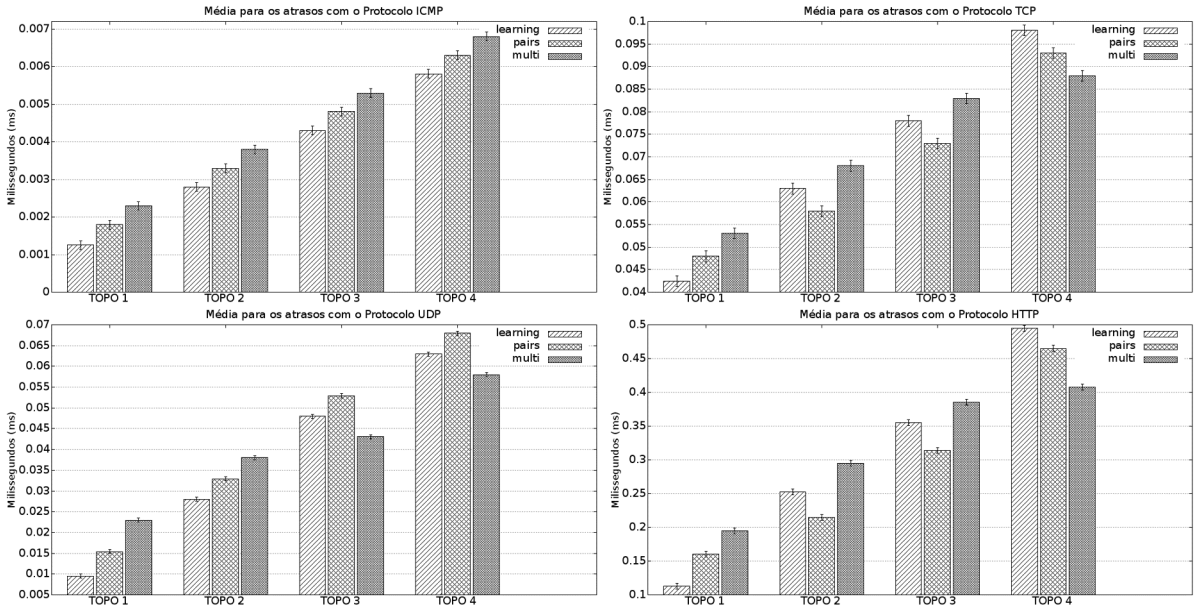
O código *l2-pairs* segue os mesmos moldes do *l2-learning* sendo em determinados casos mais eficiente, por instalar regras baseadas apenas em endereços MAC, tratando a origem e o destino como um par de endereços l2, por conseguinte, eliminando possíveis inserções de fluxos na tabela de fluxos. Por isso o *l2-pairs* foi melhor do que *l2-learning* em alguns casos utilizando o protocolos de transporte e de aplicação.

O código *l2-multi* só obteve o melhor resultado com as topologias 3 e 4. O resultado é plausível pois o mesmo funciona como um switch de aprendizagem básico, a diferença está em ambientes com múltiplos switches, apesar que tal característica só se destacou com o quarto cenário de topologias testado. Isso se explica pois as demais topologias de teste possuem poucos switches, inclusive a primeira topologia possui apenas 1 switch e a segunda topologia possui 2 switches. Já a terceira topologia, que possui 3 switches (porém apenas 1 deles é OpenFlow), obteve melhor performance com a regra dinâmica *l2-multi* com a carga de pacotes maior e com os protocolos TCP e HTTP.

Os valores apresentados a seguir são um comparativo do desempenho dos três conjuntos de regras do POX com a carga baixa de pacotes (100 pacotes), variando a topologia e os protocolos. Os resultados foram utilizados para o treinamento e validação da RNA.

A Figura 17 mostra qual conjunto de regras obteve a melhor performance para a quantidade de pacotes baixa, alicerçados na média e nos intervalos de confiança para os atrasos em milissegundos.

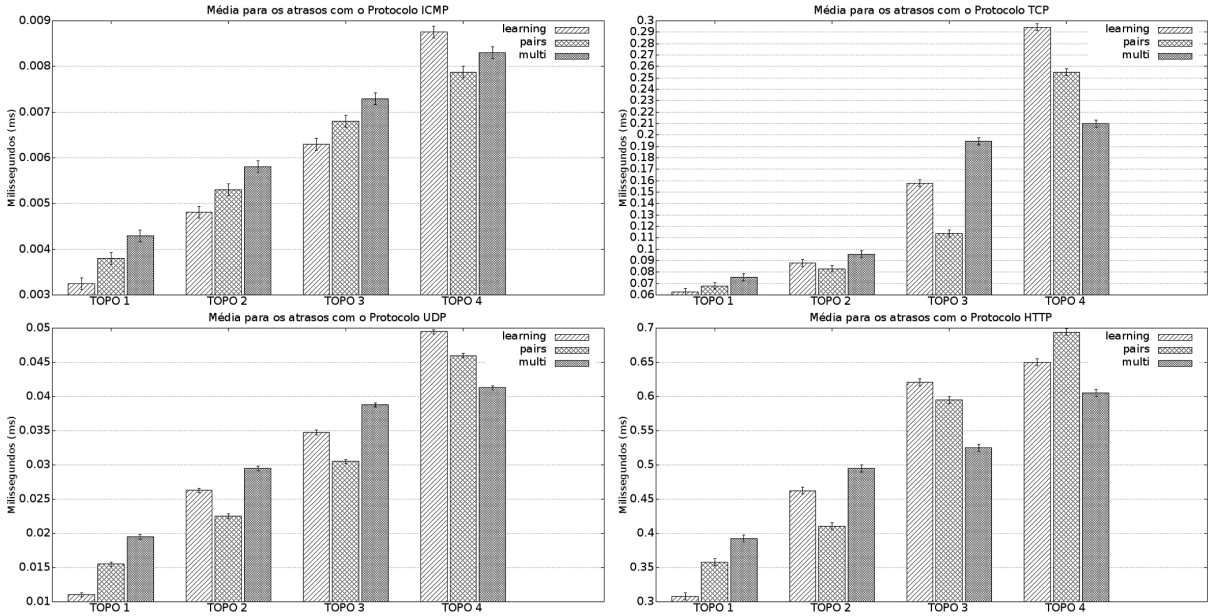
Figura 17 – Latência para quantidade de pacotes baixa - *match por porta*



Fonte: (PRÓPRIO AUTOR)

A Figura 18 mostra qual conjunto de regras obteve a melhor performance para a carga de 1.000 pacotes, baseados na média e nos intervalos de confiança para os atrasos.

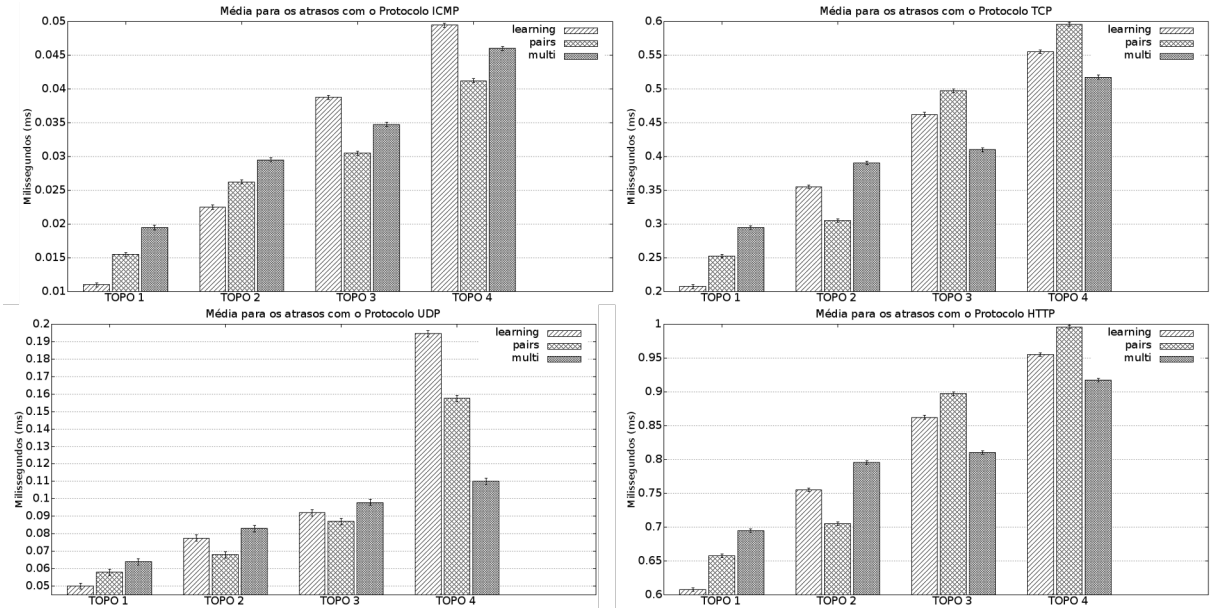
Figura 18 – Latência para quantidade de pacotes média - *match por porta*



Fonte: (PRÓPRIO AUTOR)

E por fim, a Figura 19 mostra qual conjunto de regras obteve a melhor performance a carga de 10.000 pacotes, também baseados na média e nos intervalos de confiança para os atrasos.

Figura 19 – Latência para quantidade de pacotes alta - *match por porta*



Fonte: (PRÓPRIO AUTOR)

Em vista dos valores apresentados nos resultados pode-se afirmar, por exemplo, que para um número pequeno de pacotes os atrasos médios atingiram menores valores para o protocolo ICMP, seguido do protocolo UDP, depois o TCP e por último o HTTP. A topologia 4, com a rede em multicamadas, foi a mais lenta. Já os conjuntos de regras dinâmicas variaram de acordo com a topologia e com o protocolo. Por exemplo, para o protocolo UDP, o *l2-learning.py* foi melhor nas topologias 1 e 2, já o *l2-multi.py* alcançou melhores resultados nas topologias 3 e 4. Comparando os valores gerados outras afirmações, como esta, podem surgir combinando os resultados de forma parecida. Fundado nessa linha de raciocínio e no banco de dados gerado pelos resultados dos experimentos, foi estipulada a base de treinamento da RNA.

4.1.2 Testes com regra estática do subconjunto de *match* por MAC

O mesmo procedimento realizado na subseção anterior foi repetido, porém agora a combinação das 3 regras dinâmicas foi com a regra estática de *match* por MAC. O intuito de execução dos experimentos com esse cenário foi comparar o comportamento das regras dinâmicas com outra funcionalidade de encaminhamento de pacotes (*layer 2*) e observar comportamento dos switches realizando *matches* de regras estáticas. Entretanto, com os resultados, iremos observar que houve uma variação no desempenho em termos de atraso médio. Já no ranqueamento das regras dinâmicas, a *l2-pairs.py* melhorou de posição com relação à *l2-learning.py* e *l2-multi.py* em algumas combinações.

Na Tabela 7 estão listados os resultados dos intervalos de confiança para os atrasos em milissegundos, utilizando a quantidade de pacotes em 100, variando as topologias e os protocolos. Para a *l2-learning.py*, para a *l2-pairs.py* e para a *l2-multi.py* todas associadas à regra estática com conjunto de *match* por MAC.

Tabela 7 – Intervalo de confiança da latência para experimentos com carga baixa de pacotes (em ms) - *match* por MAC

Qtde Pkt	**Topo**	**Protocolo**	**learning**	**pairs**	**multi**
100	1	ICMP	0,003194 - 0,003208	0,003691 - 0,003705	0,004190 - 0,004204
100	2	ICMP	0,004696 - 0,004710	0,005194 - 0,005208	0,005696 - 0,005711
100	3	ICMP	0,006692 - 0,006706	0,006198 - 0,006212	0,007192 - 0,007206
100	4	ICMP	0,007688 - 0,007702	0,008192 - 0,008206	0,008692 - 0,008706
100	1	UDP	0,011906 - 0,012048	0,016944 - 0,017086	0,021932 - 0,022075
100	2	UDP	0,026969 - 0,027113	0,031921 - 0,032063	0,036920 - 0,037063
100	3	UDP	0,046958 - 0,047104	0,051927 - 0,052068	0,041925 - 0,042069
100	4	UDP	0,061991 - 0,062136	0,066896 - 0,067038	0,056949 - 0,057095
100	1	TCP	0,061910 - 0,062055	0,066934 - 0,067077	0,074891 - 0,075245
100	2	TCP	0,087900 - 0,088046	0,082927 - 0,083070	0,095209 - 0,095521
100	3	TCP	0,157198 - 0,157734	0,113756 - 0,114741	0,194449 - 0,194768
100	4	TCP	0,294356 - 0,294747	0,254930 - 0,255287	0,209651 - 0,210364
100	1	HTTP	0,204963 - 0,205334	0,252363 - 0,252544	0,291376 - 0,291553
100	2	HTTP	0,359824 - 0,360167	0,307258 - 0,307817	0,394250 - 0,394573
100	3	HTTP	0,460982 - 0,461056	0,409697 - 0,410426	0,497971 - 0,498043
100	4	HTTP	0,596338 - 0,596518	0,554836 - 0,555197	0,509688 - 0,510400

Com a repetição dos experimentos porém com outra regra estática em funcionamento, o conjunto de regras dinâmicas com melhor desempenho, também, alterou de acordo com a topologia e com protocolo utilizados, independente da quantidade de pacotes. De forma geral, a resposta de cada experimento seguiu os moldes das funções pertinentes ao seu respectivo conjunto de regras. O código *l2-pairs* por possuir características e funcionalidades associadas ao endereço MAC, ultrapassou a regra *l2-learning* algumas posições no *ranking*. A regra *l2-multi* também perdeu posições para a regra *l2-pairs*, quando testado com a topologia 3.

Na sequência a carga de pacotes foi alterada de 100 para 1.000. Entretanto, na Tabela 8 estão listados os resultados dos intervalos de confiança para os atrasos em milissegundos, utilizando a quantidade de pacotes em 1.000, variando as topologias e os protocolos. Para a *l2-learning.py*, para a *l2-pairs.py* e para a *l2-multi.py* todas associadas à regra estática com conjunto de *match* por MAC.

Tabela 8 – Intervalo de confiança da latência para experimentos com carga média de pacotes (em ms) - *match* por MAC

Qtde Pkt	Topo	Protocolo	learning	pairs	multi
1000	1	ICMP	0,003695 - 0,003703	0,004195 - 0,004203	0,004693 - 0,004702
1000	2	ICMP	0,005196 - 0,005205	0,005698 - 0,005706	0,006198 - 0,006206
1000	3	ICMP	0,007193 - 0,007201	0,006697 - 0,006705	0,007697 - 0,007706
1000	4	ICMP	0,009201 - 0,009209	0,008120 - 0,008447	0,008694 - 0,008702
1000	1	UDP	0,020985 - 0,021026	0,025495 - 0,025515	0,029485 - 0,029505
1000	2	UDP	0,036245 - 0,036255	0,032494 - 0,032515	0,039482 - 0,039502
1000	3	UDP	0,044747 - 0,044757	0,040481 - 0,040501	0,048741 - 0,048751
1000	4	UDP	0,059428 - 0,059450	0,055980 - 0,056001	0,051225 - 0,051277
1000	1	TCP	0,077971 - 0,078053	0,072471 - 0,072575	0,082991 - 0,083075
1000	2	TCP	0,095348 - 0,095534	0,087977 - 0,088059	0,113702 - 0,114280
1000	3	TCP	0,157430 - 0,157739	0,209893 - 0,210306	0,194390 - 0,194578
1000	4	TCP	0,294313 - 0,294540	0,312199 - 0,312718	0,254919 - 0,255124
1000	1	HTTP	0,457298 - 0,457609	0,407368 - 0,407678	0,492366 - 0,492672
1000	2	HTTP	0,562443 - 0,562548	0,509778 - 0,510193	0,594762 - 0,594967
1000	3	HTTP	0,624907 - 0,625115	0,694871 - 0,695076	0,654934 - 0,655140
1000	4	HTTP	0,749882 - 0,750091	0,794357 - 0,794582	0,704940 - 0,705148

A Tabela 9 lista os resultados dos intervalos de confiança para os atrasos em milissegundos, utilizando a quantidade de pacotes em 10.000, variando as topologias e os protocolos. Isso para a *l2-learning.py*, para a *l2-pairs.py* e para a *l2-multi.py* todas associadas à regra estática com conjunto de *match* por MAC.

Tabela 9 – Intervalo de confiança da latência para experimentos com carga alta de pacotes (em ms) - *match* por MAC

Qtde Pkt	Topo	Protocolo	learning	pairs	multi
10000	1	ICMP	0,020985 - 0,021008	0,025493 - 0,025505	0,029493 - 0,029504
10000	2	ICMP	0,036248 - 0,036253	0,032496 - 0,032507	0,039489 - 0,039501
10000	3	ICMP	0,048747 - 0,048752	0,040491 - 0,040503	0,044746 - 0,044752
10000	4	ICMP	0,059435 - 0,059447	0,051232 - 0,051260	0,055992 - 0,056003
10000	1	UDP	0,064985 - 0,065076	0,067991 - 0,068036	0,073977 - 0,074000
10000	2	UDP	0,087460 - 0,087516	0,077979 - 0,078025	0,092968 - 0,093014
10000	3	UDP	0,109982 - 0,110209	0,096990 - 0,097013	0,189419 - 0,189633
10000	4	UDP	0,294443 - 0,294569	0,257399 - 0,257569	0,209975 - 0,210201
10000	1	TCP	0,352433 - 0,352603	0,307475 - 0,307644	0,394861 - 0,394973
10000	2	TCP	0,454904 - 0,455016	0,404964 - 0,405077	0,490469 - 0,490503
10000	3	TCP	0,509894 - 0,510120	0,597436 - 0,597492	0,562474 - 0,562531
10000	4	TCP	0,654935 - 0,655048	0,695479 - 0,695581	0,617453 - 0,617510
10000	1	HTTP	0,657454 - 0,657511	0,607335 - 0,607505	0,694932 - 0,695043
10000	2	HTTP	0,754916 - 0,755030	0,704926 - 0,705038	0,795503 - 0,795559
10000	3	HTTP	0,809986 - 0,810212	0,897429 - 0,897485	0,862472 - 0,862529
10000	4	HTTP	0,954997 - 0,955110	0,995451 - 0,995553	0,917462 - 0,917519

Os resultados do ranqueamento para a quantidade de 10.000 pacotes foram iguais aos dos experimentos com 1.000 pacotes, com exceção das topologia 2 e 3 para o protocolo ICMP, onde a *l2-pairs.py* obteve a melhor performance. A *l2-multi.py* e a *l2-learning.py* alternaram as segundas e terceiras posições.

Com os testes com uma nova regra estática, em muitos casos, a resposta de cada experimento também seguiu as funcionalidades pertinentes ao seu respectivo conjunto de regras, porém com valores de atraso médio maiores demonstrando que a regra de *match* por MAC é mais lenta do que a regra de *match* por porta.

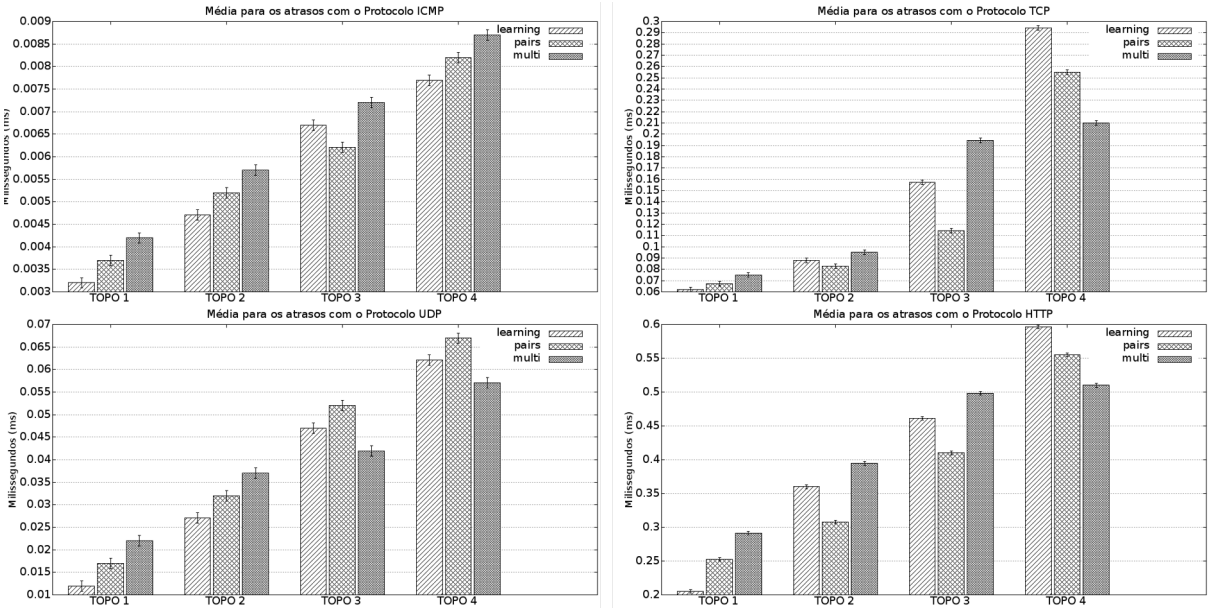
O código *l2-pairs* segue os mesmos moldes do *l2-learning* sendo em determinados casos mais eficiente, por instalar regras baseadas apenas em endereços MAC, tratando a origem e o destino como um par de endereços l2 e, com isso, eliminando possíveis inserções de fluxos na tabela de fluxos. Por isso, a *l2-pairs* foi melhor do que *l2-learning* em combinações, que nos testes da subseção anterior, foram completamente dominados por tal regra de aprendizagem, por exemplo com 100 pacotes ICMP.

O código *l2-multi* ainda continuou com os melhores resultados com a topologias 4. Todavia, na topologia 3 quando havia sido mais rápido com a outra regra estática, também perdeu posições para a regra *l2-pairs*.

Os valores apresentados a seguir são um comparativo do desempenho dos três conjuntos de regras do POX com a quantidade de pacotes em 100, variando a topologia e os protocolos. Os resultados foram utilizados para o treinamento e validação da RNA.

A Figura 20 mostra qual conjunto de regras obteve a melhor performance para a quantidade de pacotes baixa, com base na média e nos intervalos de confiança para os atrasos em milissegundos.

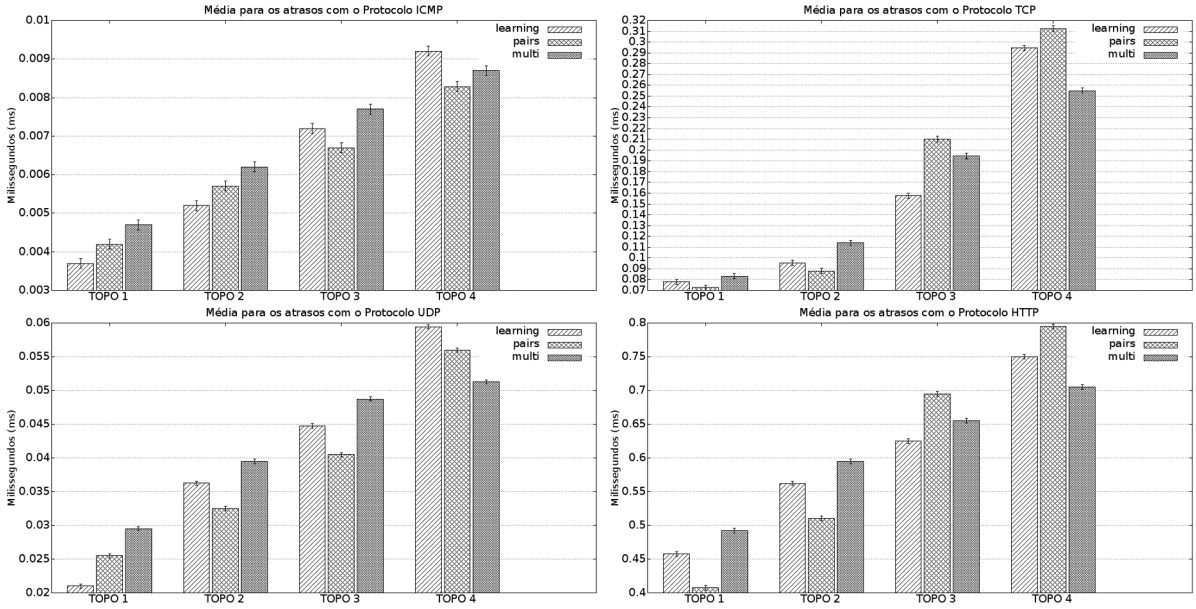
Figura 20 – Latência para quantidade de pacotes baixa - *match por MAC*



Fonte: (PRÓPRIO AUTOR)

A Figura 21 mostra qual conjunto de regras obteve a melhor performance para a carga de pacotes em 1.000, baseados na média e nos intervalos de confiança para os atrasos.

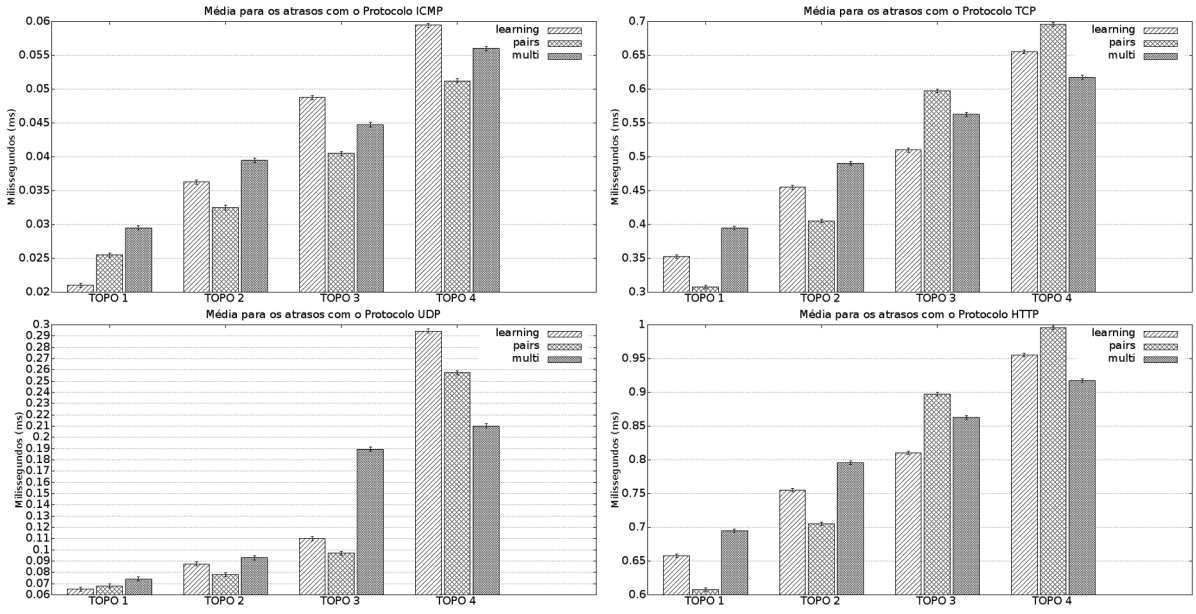
Figura 21 – Latência para quantidade de pacotes média - *match por MAC*



Fonte: (PRÓPRIO AUTOR)

E por fim, a Figura 22 mostra qual conjunto de regras obteve a melhor performance a carga de pacotes em 10.000, também baseados na média e nos intervalos de confiança para os atrasos.

Figura 22 – Latência para quantidade de pacotes alta - *match por MAC*



Fonte: (PRÓPRIO AUTOR)

Os resultados demonstraram que, diferentemente, da regra estática de *match* por porta, com a regra estática de *match* por MAC houve alteração no *ranking* quando os testes foram com 100 pacotes ICMP. Foram mantidas as posições de melhor performance para um número pequeno de pacote. Nesse caso, os atrasos médios atingiram menores valores para o protocolo ICMP, seguido do protocolo UDP, depois o TCP e por último o HTTP.

A topologia 4, com a rede em multicamadas, também foi a mais lenta. Já os conjuntos de regras dinâmicas continuaram variando de acordo com a topologia e com o protocolo. Por exemplo, para o protocolo UDP, a *l2-pairs.py* obteve um desempenho melhor com a carga maior de pacotes, já o *l2-multi.py* piorou seus resultados na topologia 3. Firme nessa linha de raciocínio e no banco de dados gerado pelos resultados dos experimentos, foi estipulada a base de treinamento da RNA.

4.1.3 Testes com o hardware-switch

Para os experimentos com o switch físico foi utilizado o hardware-switch Extreme Summit x440-48p, com sistema operacional ExtremeXOS versão 16.2.3.5 e processador Single Core CPU 500 MHz. Tal equipamento suporta, com a versão original de *firmware* a versão 1.0 e 1.1 do OpenFlow, além de suportar até 1024 regras sendo 248 no modo Openflow. Todos os testes foram feitos exclusivamente para a topologia 1, foi montada uma estrutura com um hardware switch citado na subseção 3.1 e três máquinas físicas com funções de controlador, cliente e ser-

vidor. A configuração dos computadores utilizada foi a seguinte: processador Intel Core i7-860 2.80 GHz, com 4GB de memória RAM para a máquina do cliente e processadores Intel Core i5-2400 3.1 GHz, com 8GB de memória RAM para as máquinas do controlador e do Servidor. Todos configurados com o sistema operacional Linux Debian 8.3.0.

Toda configuração da SDN, com exceção do switch OpenFlow, seguiram os mesmos detalhes dos experimentos com o virtualizador Mininet.

O objetivo dos testes com equipamento físico é fazer um comparativo com o switch virtual verificando se o tempo de execução da regras dinâmicas e de execução dos *matches* é muito discrepante, além de verificar o desempenho das mesmas regras dinâmicas e dos mesmos *matches* em switches diferentes.

A Tabela 10 lista os intervalos de confiança para os atrasos em milissegundos, utilizando a quantidade de pacotes em 100, 1.000 e 10.000, variando os protocolos e com a topologia 1 fixa. Isso para a *l2-learning.py*, para a *l2-pairs.py* e para a *l2-multi.py* também associadas à regra estática com conjunto de *match* por porta.

Tabela 10 – Intervalo de confiança da latência para experimentos com *match* por porta (em ms) - Testes com o hardware-switch

Qtde Pkt	**Topo**	**Protocolo**	**learning**	**pairs**	**multi**
100	1	ICMP	0,005471 - 0,005506	0,006479 - 0,006514	0,007475 - 0,007510
100	1	UDP	0,014651 - 0,015002	0,034750 - 0,035099	0,044593 - 0,044944
100	1	TCP	0,094830 - 0,095186	0,109665 - 0,110379	0,159793 - 0,160146
100	1	HTTP	0,262748 - 0,263113	0,304875 - 0,305234	0,362377 - 0,362559
1000	1	ICMP	0,014892 - 0,015098	0,024902 - 0,025110	0,034878 - 0,035087
1000	1	UDP	0,084877 - 0,085084	0,074894 - 0,075101	0,112208 - 0,112722
1000	1	TCP	0,207324 - 0,207635	0,254889 - 0,255096	0,294879 - 0,295082
1000	1	HTTP	0,409762 - 0,410178	0,459927 - 0,460134	0,497916 - 0,497998
10000	1	ICMP	0,109894 - 0,110120	0,154950 - 0,155063	0,194435 - 0,194538
10000	1	UDP	0,359953 - 0,360066	0,307363 - 0,307532	0,394454 - 0,394534
10000	1	TCP	0,609899 - 0,610125	0,654874 - 0,654987	0,694390 - 0,694492
10000	1	HTTP	0,812351 - 0,812636	0,857476 - 0,857533	0,891963 - 0,892031

A Tabela 11 lista os intervalos de confiança para os atrasos em milissegundos, utilizando a quantidade de pacotes em 100, 1.000 e 10.000, variando os protocolos e com a topologia 1 fixa. Isso para a *l2-learning.py*, para a *l2-pairs.py* e para a *l2-multi.py* também associadas à regra estática com conjunto de *match* por MAC.

Tabela 11 – Intervalo de confiança da latência para experimentos com *match* por MAC (em ms) - Testes com o hardware-switch

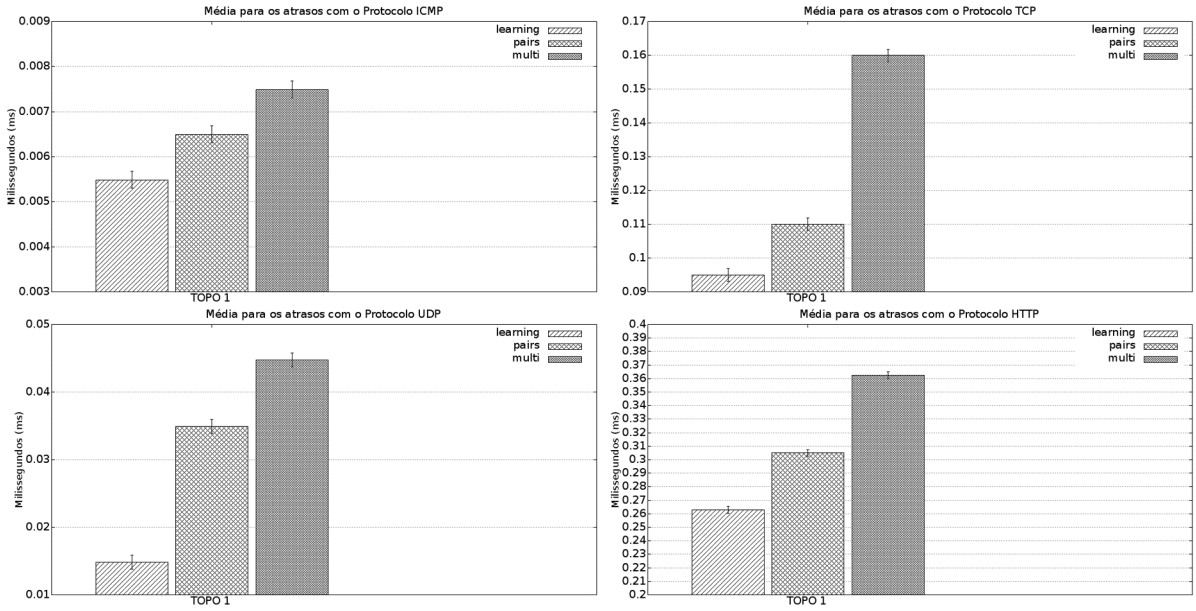
Qtde Pkt	Topo	Protocolo	learning	pairs	multi
100	1	ICMP	0,014839 - 0,015208	0,034744 - 0,035098	0,044828 - 0,045182
100	1	UDP	0,094743 - 0,095093	0,109821 - 0,11054	0,159915 - 0,160273
100	1	TCP	0,304815 - 0,305173	0,361372 - 0,361618	0,396909 - 0,397052
100	1	HTTP	0,414697 - 0,415049	0,457379 - 0,457558	0,496963 - 0,497035
1000	1	ICMP	0,074868 - 0,075075	0,089762 - 0,090176	0,112198 - 0,112716
1000	1	UDP	0,254817 - 0,255024	0,207426 - 0,207735	0,294821 - 0,295027
1000	1	TCP	0,459864 - 0,460072	0,409847 - 0,410261	0,497912 - 0,497992
1000	1	HTTP	0,551394 - 0,55154	0,504858 - 0,505065	0,590989 - 0,591072
10000	1	ICMP	0,212464 - 0,212521	0,257478 - 0,257535	0,2925 - 0,292555
10000	1	UDP	0,362452 - 0,362508	0,309844 - 0,310072	0,397469 - 0,397502
10000	1	TCP	0,652922 - 0,653035	0,607444 - 0,607613	0,694493 - 0,694527
10000	1	HTTP	0,851496 - 0,85153	0,804931 - 0,805046	0,891962 - 0,892029

Os valores apresentados a seguir são um comparativo do desempenho dos três conjuntos de regras dinâmicas do POX com a quantidade de pacotes em 100, variando a topologia e os protocolos. Os resultados foram utilizados para o treinamento e validação da RNA.

Vale ressaltar que as 3 regras dinâmicas estão associadas à regra estática com conjunto de *match* por porta e também à regra estática com conjunto de *match* por MAC.

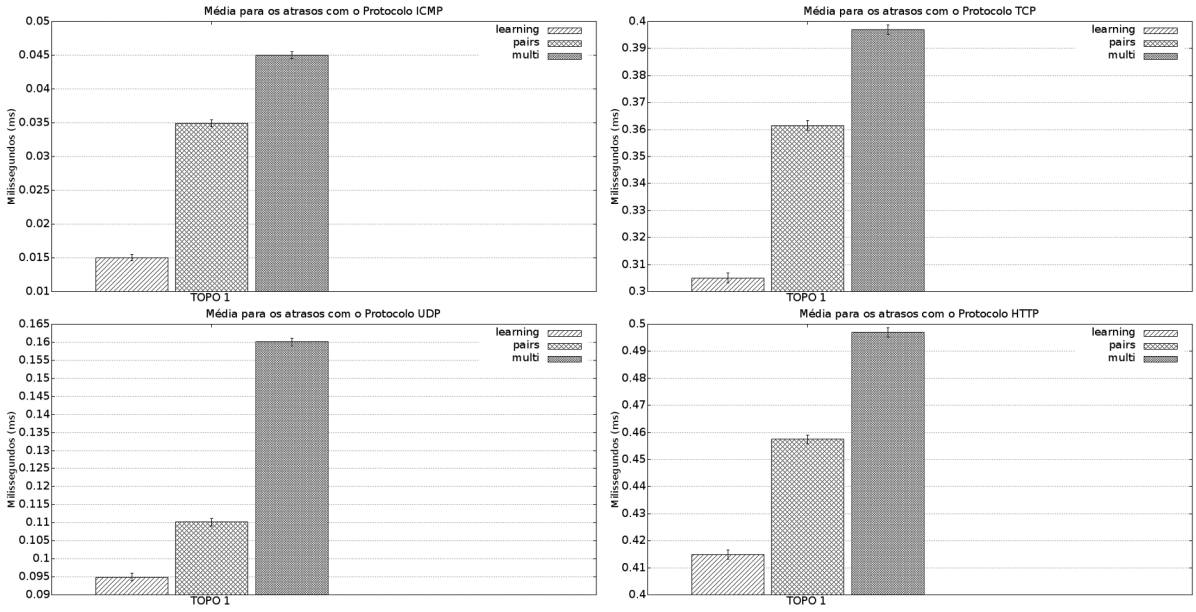
Observando as Figuras 23 e 24 é possível visualizar qual conjunto de regras obteve a melhor performance para a quantidade de pacotes baixa, baseados na média e nos intervalos de confiança para os atrasos em milissegundos. Para a regra estática com conjunto de *match* por porta e para regra estática de *match* por MAC, respectivamente.

Figura 23 – Latência para quantidade de pacotes baixa - *match* por porta (Testes com o hardware-switch)



Fonte: (PRÓPRIO AUTOR)

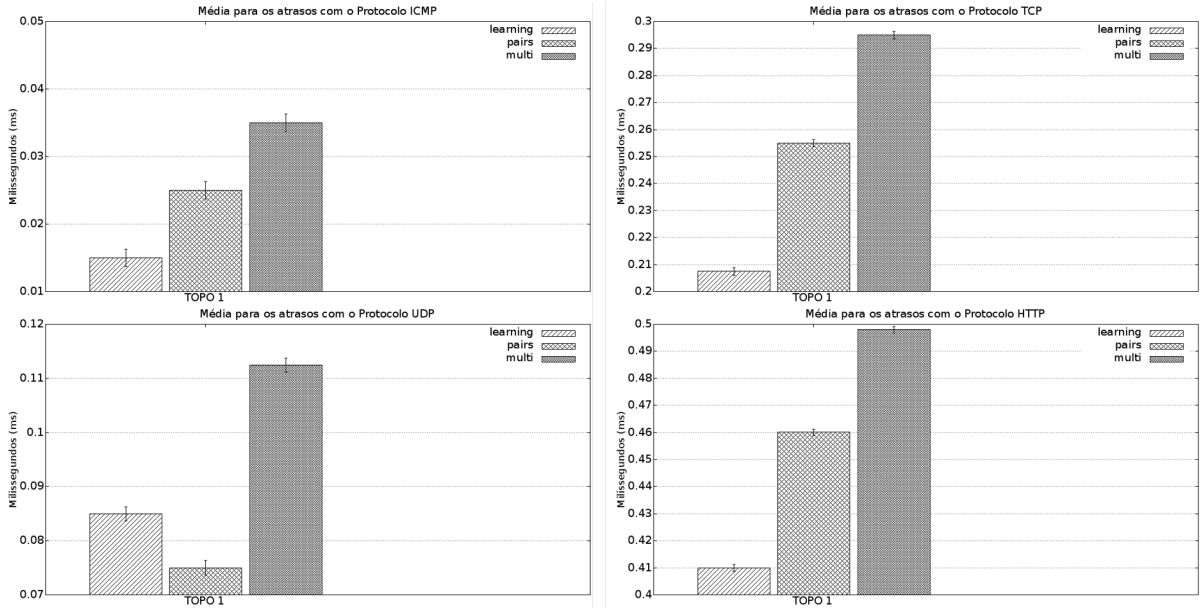
Figura 24 – Latência para quantidade de pacotes baixa - *match* por MAC (Testes com o hardware-switch)



Fonte: (PRÓPRIO AUTOR)

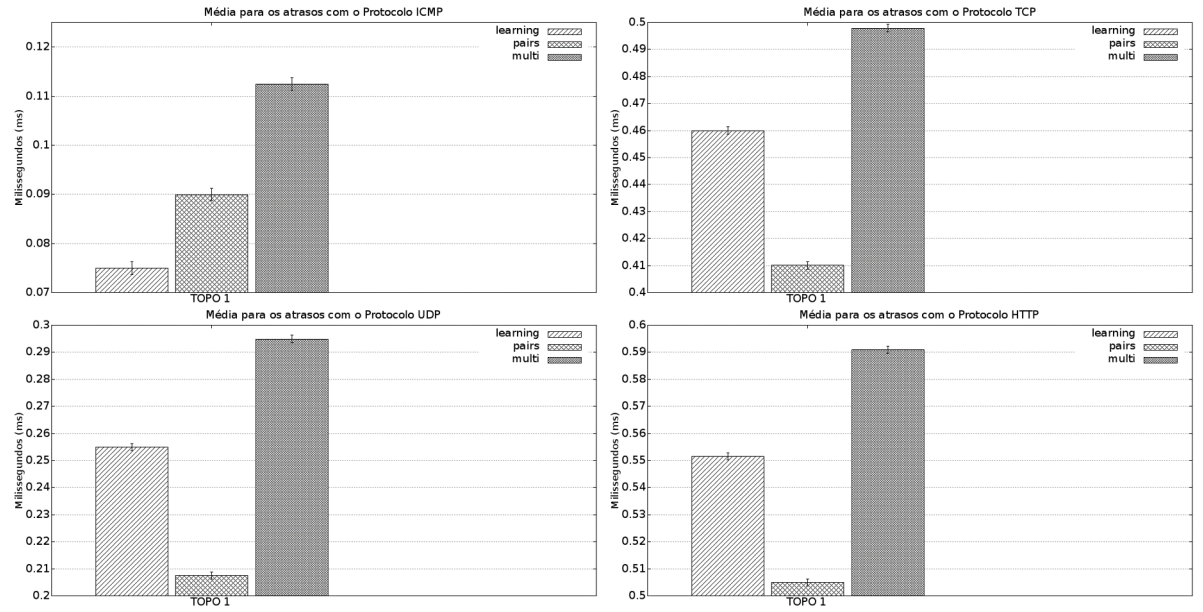
As Figuras 25 e 26 mostram qual conjunto de regras obteve a melhor performance para a carga de pacotes em 1.000, embasados na média e nos intervalos de confiança para os atrasos. Para a regra estática com conjunto de *match* por porta e para regra estática de *match* por MAC, respectivamente.

Figura 25 – Latência para quantidade de pacotes média - *match* por porta (Testes com o hardware-switch)



Fonte: (PRÓPRIO AUTOR)

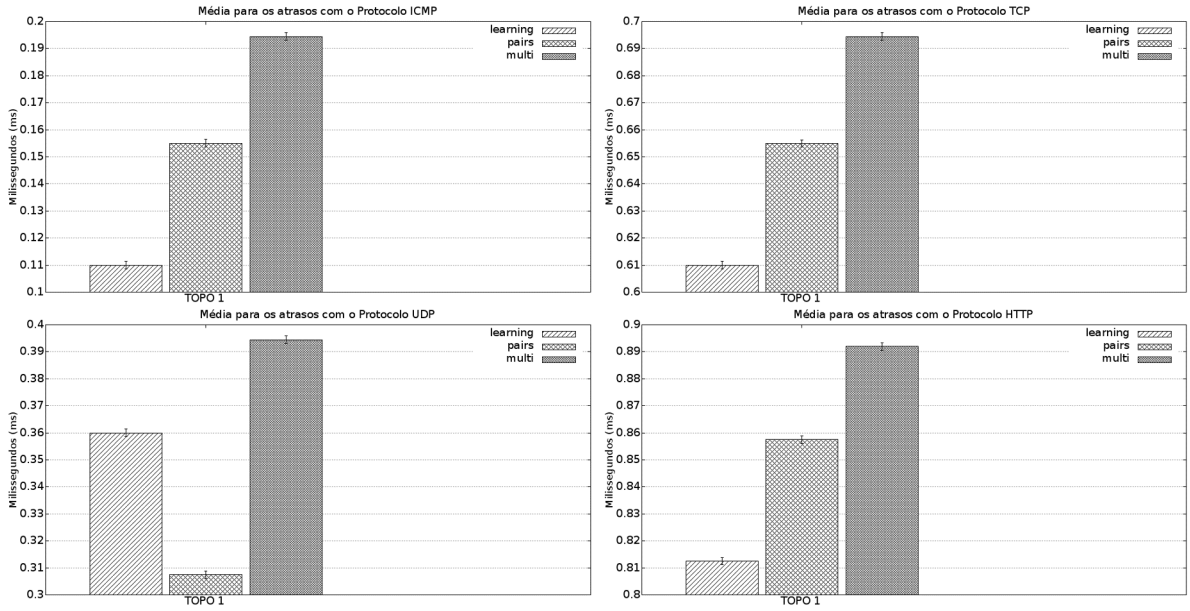
Figura 26 – Latência para quantidade de pacotes média - *match* por MAC (Testes com o hardware-switch)



Fonte: (PRÓPRIO AUTOR)

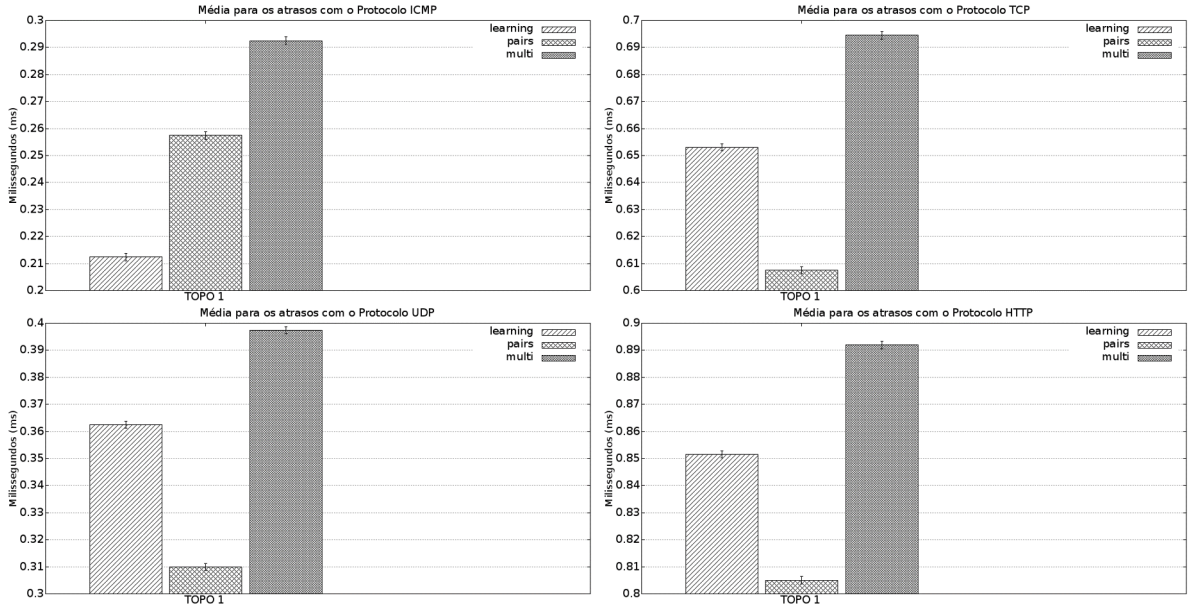
Por último, a Figuras 27 e 28 mostram qual conjunto de regras obteve a melhor performance a carga de pacotes em 10.000, também baseados na média e nos intervalos de confiança para os atrasos. E também para a regra estática com conjunto de *match* por porta e para regra estática de *match* por MAC, respectivamente.

Figura 27 – Latência para quantidade de pacotes alta -*match* por porta (Testes com o hardware-switch)



Fonte: (PRÓPRIO AUTOR)

Figura 28 – Latência para quantidade de pacotes alta -*match* por MAC (Testes com o hardware-switch)



Fonte: (PRÓPRIO AUTOR)

Os resultados foram previsíveis no quesito atraso médio, nos quais o hardware-switch obteve valores mais altos do que o software-switch rodando através do virtualizador de rede Mininet. Isso se explica pelo fato que o switch virtual OvS rodando dentro de um simulador não possuir grandes problemas com relação à distância de barramento percorrida. Com o switch físico existem todos os componentes de hardware envolvidos no enlace, desde a interface controladora de rede da máquina de origem, passando pelas interfaces do switch, até a interface

controladora de rede da máquina de destino (RUCKERT et al., 2014). Já com o uso de simulador, todos esses componentes estão embutidos em uma só máquina e o enlace é todo feito de forma virtual. Para exemplificar basta observar o experimento com a carga de 10.000 pacotes e com o protocolo ICMP, no switch virtual os valores de atraso médio variaram de 0,01 e 0,05 milissegundos, já com o switch físico os valores ficaram entre 0,1 e 0,2 milissegundos.

Na análise dos resultados como um todo o ranqueamento variou nas combinações envolvendo o protocolo UDP. Por exemplo, para as cargas com 1.000 e 10.000 pacotes com o protocolo UDP a *l2-learning.py* perdeu sua primeira posição para a regra *l2-pairs.py*, rodando com a regra estática com conjunto de *match* por porta. O mesmo comportamento pôde ser observado rodando com a regra estática com conjunto de *match* por MAC.

4.1.4 Modelo de avaliação das regras

Cumprindo as etapas geradas pelos cenários citados na seção 3, todas as estatísticas de fluxo, especificamente atraso, e os resultados coletados nas subseções anteriores são armazenados num arquivo texto (.txt). Em seguida, por experimento, é feito um ranqueamento com base nos atrasos médios gerados. A Tabela 12 representa a configuração de um dos experimentos, que utiliza os 3 conjuntos de regras dinâmicas (*learning*, *pairs* e *multi*) com um regra estática para subconjunto de *match* por porta. As cores verdes indicam melhor performance, seguida das cores amarelas e a pior performance é representada pelas cores vermelhas.

Dadas as 4 topologias, os 4 protocolos e as 3 quantidades de pacotes é possível avaliar qual das 3 regras dinâmicas tem a melhor performance. Se o controlador SDN (POX) estiver com a regra *pairs* como regra vigente e encontrar uma situação em que a quantidade de pacotes TCP for alta, com uma topologia em árvore, com 1 swtich openflow e 2 swtiches convencionais (topologia 3), é possível dizer que a regra atual é a pior opção para o cenário dentre as 3 avaliadas. O processo explanado nesse experimento serve de modelo para os demais experimentos, pois todos seguiram o mesmo padrão de configuração.

Tabela 12 – Modelo de ranqueamento para um dos experimentos

Regra dinâmica + regra estática (match por porta)										
l2_Learning				**l2_pairs**				**l2_multi**		
Qtd Pkt	Protocolo	Topologia		Qtd Pkt	Protocolo	Topologia		Qtd Pkt	Protocolo	Topologia
Experimento 1 - CARGA ALTA										
10000	ICMP	TOPO 1		10000	ICMP	TOPO 1		10000	ICMP	TOPO 1
10000	ICMP	TOPO 2		10000	ICMP	TOPO 2		10000	ICMP	TOPO 2
10000	ICMP	TOPO 3		10000	ICMP	TOPO 3		10000	ICMP	TOPO 3
10000	ICMP	TOPO 4		10000	ICMP	TOPO 4		10000	ICMP	TOPO 4
10000	UDP	TOPO 1		10000	UDP	TOPO 1		10000	UDP	TOPO 1
10000	UDP	TOPO 2		10000	UDP	TOPO 2		10000	UDP	TOPO 2
10000	UDP	TOPO 3		10000	UDP	TOPO 3		10000	UDP	TOPO 3
10000	UDP	TOPO 4		10000	UDP	TOPO 4		10000	UDP	TOPO 4
10000	TCP	TOPO 1		10000	TCP	TOPO 1		10000	TCP	TOPO 1
10000	TCP	TOPO 2		10000	TCP	TOPO 2		10000	TCP	TOPO 2
10000	TCP	TOPO 3		10000	TCP	TOPO 3		10000	TCP	TOPO 3
10000	TCP	TOPO 4		10000	TCP	TOPO 4		10000	TCP	TOPO 4
10000	HTTP	TOPO 1		10000	HTTP	TOPO 1		10000	HTTP	TOPO 1
10000	HTTP	TOPO 2		10000	HTTP	TOPO 2		10000	HTTP	TOPO 2
10000	HTTP	TOPO 3		10000	HTTP	TOPO 3		10000	HTTP	TOPO 3
10000	HTTP	TOPO 4		10000	HTTP	TOPO 4		10000	HTTP	TOPO 4
Experimento 2 - CARGA MÉDIA										
1000	ICMP	TOPO 1		1000	ICMP	TOPO 1		1000	ICMP	TOPO 1
1000	ICMP	TOPO 2		1000	ICMP	TOPO 2		1000	ICMP	TOPO 2
1000	ICMP	TOPO 3		1000	ICMP	TOPO 3		1000	ICMP	TOPO 3
1000	ICMP	TOPO 4		1000	ICMP	TOPO 4		1000	ICMP	TOPO 4
1000	UDP	TOPO 1		1000	UDP	TOPO 1		1000	UDP	TOPO 1
1000	UDP	TOPO 2		1000	UDP	TOPO 2		1000	UDP	TOPO 2
1000	UDP	TOPO 3		1000	UDP	TOPO 3		1000	UDP	TOPO 3
1000	UDP	TOPO 4		1000	UDP	TOPO 4		1000	UDP	TOPO 4
1000	TCP	TOPO 1		1000	TCP	TOPO 1		1000	TCP	TOPO 1
1000	TCP	TOPO 2		1000	TCP	TOPO 2		1000	TCP	TOPO 2
1000	TCP	TOPO 3		1000	TCP	TOPO 3		1000	TCP	TOPO 3
1000	TCP	TOPO 4		1000	TCP	TOPO 4		1000	TCP	TOPO 4
1000	HTTP	TOPO 1		1000	HTTP	TOPO 1		1000	HTTP	TOPO 1
1000	HTTP	TOPO 2		1000	HTTP	TOPO 2		1000	HTTP	TOPO 2
1000	HTTP	TOPO 3		1000	HTTP	TOPO 3		1000	HTTP	TOPO 3
1000	HTTP	TOPO 4		1000	HTTP	TOPO 4		1000	HTTP	TOPO 4
Experimento 3 - CARGA BAIXA										
100	ICMP	TOPO 1		100	ICMP	TOPO 1		100	ICMP	TOPO 1
100	ICMP	TOPO 2		100	ICMP	TOPO 2		100	ICMP	TOPO 2
100	ICMP	TOPO 3		100	ICMP	TOPO 3		100	ICMP	TOPO 3
100	ICMP	TOPO 4		100	ICMP	TOPO 4		100	ICMP	TOPO 4
100	UDP	TOPO 1		100	UDP	TOPO 1		100	UDP	TOPO 1
100	UDP	TOPO 2		100	UDP	TOPO 2		100	UDP	TOPO 2
100	UDP	TOPO 3		100	UDP	TOPO 3		100	UDP	TOPO 3
100	UDP	TOPO 4		100	UDP	TOPO 4		100	UDP	TOPO 4
100	TCP	TOPO 1		100	TCP	TOPO 1		100	TCP	TOPO 1
100	TCP	TOPO 2		100	TCP	TOPO 2		100	TCP	TOPO 2
100	TCP	TOPO 3		100	TCP	TOPO 3		100	TCP	TOPO 3
100	TCP	TOPO 4		100	TCP	TOPO 4		100	TCP	TOPO 4
100	HTTP	TOPO 1		100	HTTP	TOPO 1		100	HTTP	TOPO 1
100	HTTP	TOPO 2		100	HTTP	TOPO 2		100	HTTP	TOPO 2
100	HTTP	TOPO 3		100	HTTP	TOPO 3		100	HTTP	TOPO 3
100	HTTP	TOPO 4		100	HTTP	TOPO 4		100	HTTP	TOPO 4

4.2 Resposta da Rede Neural Artificial

Além dos resultados obtidos com a RNA, nessa subseção optou-se em inserir detalhes metodológicos e experimentais que se ausentaram da seção de metodologia pelo motivo de que nessa parte do trabalho o leitor já possui o embasamento necessário para interpretar, de modo mais completo, tais configurações. Outro motivo é que os detalhes citados, sendo descritos nessa parte do texto, respeitam uma sequência cronológica de configurações da RNA.

A seguir o comportamento da RNA e suas particularidades são demonstrados.

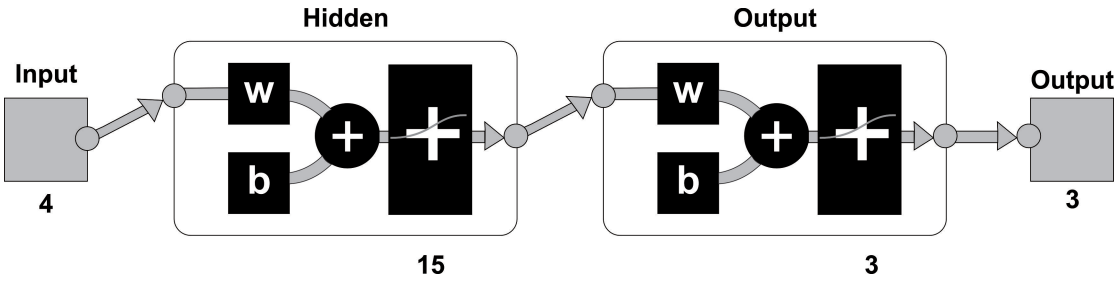
4.2.1 Configuração da Rede Neural Artificial

Foram testadas as duas topologias, citadas na início da subseção 3.7, em experimentos com a RNA. A topologia com a presença de apenas uma camada oculta com 15 neurônios e três saídas foi a melhor opção dentre as duas experimentadas, pois suas três saídas trouxeram melhor resposta aos testes que variaram quantidade de pacotes e vazão da rede em três níveis.

O software para a resolução do problema utilizando redes neurais artificiais foi o Matlab (MATLAB, 2018). O Matlab foi escolhido por ser um software científico computacional, que apresenta um *toolbox* chamado NPRTOOL com funções voltadas especialmente para classificação de padrões em redes neurais e apresenta, também, o comando *patternnet* com funcionalidades complementares para o mesmo problema (HAGAN et al., 1996).

A rede neural artificial foi configurada no Matlab conforme mostra a Figura 29.

Figura 29 – Configuração da RNA



Fonte: (PRÓPRIO AUTOR)

As entradas da RNA são os resultados das medições constituídas nas métricas de atraso e quantidade de pacotes, além de valores referentes às topologias e aos protocolos.

As três saídas geram uma combinação de três números inteiros que podem ser apenas 0 (zero) ou 1 (um). Inicialmente a rede neural artificial classifica três tipos conjuntos de regras

do controlador, porém com essa mesma configuração existe a possibilidade de classificação automática de até oito tipos de regra, dependendo do treinamento.

A padronização para a saída, a qual representa o conjunto de regras a ser aplicado, ficou definida de acordo com a Tabela 13. As três saídas geram uma combinação de três números inteiros que podem ser apenas 0 (zero) ou 1 (um). Inicialmente a rede neural artificial classifica três tipos conjuntos de regras dinâmicas do controlador, porém com essa configuração existe a possibilidade de classificação automática de até oito tipos de regra, dependendo do treinamento, ou seja, 2^3 resulta em oito combinações possíveis para três bits zero ou três bits um (100, 010 e 001 são 3 das 8 possíveis).

Tabela 13 – Padronização das saídas da rede neural artificial

Tipo de regra	y1	y2	y3
l2-learning	1	0	0
l2-pairs	0	1	0
l2-multi	0	0	1

4.2.2 Treinamento da RNA

Para o treinamento da rede neural, foi adotado o tipo supervisionado, no qual a rede possui um componente que verifica os dados reais e os confere com os dados de saída do aprendizado retornando o valor de um erro para a rede. Foi utilizada uma rede perceptron multicamadas - MLPs, para esse tipo de configuração o algoritmo de treinamento indicado é o *backpropagation* do tipo supervisionado que acontece em duas etapas: a primeira em que é feita a recepção dos dados indo em direção a camada de saída e a segunda etapa é quando ocorre a tramitação inversa dos dados, onde é calculado o erro da rede e alteração dos pesos (SILVA; SPATTI; FLAUZINO, 2010).

Os dados utilizados para o treinamento, teste e validação da rede neural foram coletados dos experimentos com redes definidas por software baseadas no protocolo OpenFlow citados nesta seção.

O treinamento foi realizado separadamente por quantidade de pacotes, isto é, com a quantidade de pacotes configurada em 100, variou-se as 4 topologias, variou-se os quatro protocolos e variou-se as 3 regras dinâmicas. O treinamento foi feito da mesma forma para as cargas de pacotes fixadas em 1.000 e 10.000. Para mais, o treinamento também foi realizado para a Topologia 1 com os resultados dos testes com o hardware-switch (também variando a quantidade de pacotes em 100, 1.000 e 10.000).

Para cada uma das opções do processo supracitado, foi estipulada e realizada a rotina de experimentos a seguir. Primeiramente os dados gerados foram misturados, 10% desses valores foram utilizados como amostras de treinamento, teste e validação da RNA. O percentual de 10% foi adotado pois, conforme mencionado na subseção de carga e métricas, a taxa de envio gerada pela operação de *flowstats* é de 10 pacotes por segundo. Por sua vez, esse conjunto de amostras foi separado em 70% das amostras para treinamento, 15% das amostras para teste e 15% das amostras para validação. A rotina realizada para *traces* com a quantidade de pacotes menor, foi espelhada para os experimentos com *traces* que incluíam maior carga de pacotes.

Primeiramente segue a configuração do treinamento para os testes com o Mininet:

- Com a quantidade de pacotes fixada em 100 por experimento, uma base de dados de 4.800 linhas com valores de atraso foi gerada, ou seja, 4.800 amostras. Depois de todas misturadas, 480 amostras foram utilizadas para o treinamento como um todo, dentre elas foram utilizadas 72 amostras para validação das 144 amostras que representam os 30% destinados à teste e validação juntos.

- Para a quantidade de pacotes fixada em 1.000 por experimento, uma base de dados de 48.000 linhas com valores de atraso foi gerada, ou seja, 48.000 amostras. Depois de todas misturadas, 4.800 amostras foram utilizadas para o treinamento como um todo, dentre elas foram utilizadas 720 amostras para validação das 1.440 amostras que representam os 30% destinados à teste e validação juntos.

- E para a quantidade de pacotes fixada em 10.000 por experimento, uma base de dados de 480.000 linhas com valores de atraso foi gerada, ou seja, 480.000 amostras. Depois de todas misturadas, 48.000 amostras foram utilizadas para o treinamento como um todo, dentre elas foram utilizadas 7.200 amostras para validação das 14.400 amostras que representam os 30% destinados à teste e validação juntos.

Agora a configuração do treinamento para os testes com o hardware-switch:

- Com a quantidade de pacotes fixada em 100 por experimento, uma base de dados de 1.200 linhas com valores de atraso foi gerada, ou seja, 1.200 amostras. Depois de todas misturadas, 120 amostras foram utilizadas para o treinamento como um todo, dentre elas foram utilizadas 18 amostras para validação das 36 amostras que representam os 30% destinados à teste e validação juntos.

- Para a quantidade de pacotes fixada em 1.000 por experimento, uma base de dados de 12.000 linhas com valores de atraso foi gerada, ou seja, 12.000 amostras. Depois de todas misturadas, 1.200 amostras foram utilizadas para o treinamento como um todo, dentre elas foram utilizadas 180 amostras para validação das 360 amostras que representam os 30% destinados à teste e validação juntos.

- E, por fim, para a quantidade de pacotes fixada em 10.000 por experimento, uma base de dados de 120.000 linhas com valores de atraso foi gerada, ou seja, 120.000 amostras. Depois de todas misturadas, 12.000 amostras foram utilizadas para o treinamento como um todo, dentre elas foram utilizadas 1.800 amostras para validação das 3.600 amostras que representam os 30% destinados à teste e validação juntos.

A validação da rede, para os experimentos com o Mininet e com a quantidade de pacotes em 100, foi feita aplicando um conjunto de teste fornecido na Tabela 14, sobrescrevendo 72 das 144 amostras iniciais de teste e validação (30% das 480 amostras). Isso feito, separadamente, por experimentos com carga de pacotes (100, 1.000 e 10.000). Logo, para os experimentos com 1.000 pacotes foi utilizada tabela com 720 linhas e para os experimentos com 10.000 pacotes foi utilizada tabela com 7.200 linhas.

Tabela 14 – Conjunto de padrões de Validação - Experimentos com o Mininet e com a quantidade de pacotes em 100

	Qtde Pkt	**Atraso**	**Topologia**	**Protocolo**	**Saídas Desejadas**			
Amostra	**x1**	**x2**	**x3**	**x4**	**d1**	**d2**	**d3**	**C. Regras**
1	100	0,011749	1	UDP	0	1	0	l2_pairs
2	100	0,074133	3	TCP	0	1	0	l2_pairs
3	100	0,494832	4	HTTP	1	0	0	l2_learning
4	100	0,337082	3	HTTP	0	1	0	l2_pairs
5	100	0,033505	2	UDP	0	1	0	l2_pairs
6	100	0,413986	4	HTTP	0	0	1	l2_multi
7	100	0,286402	2	HTTP	0	0	1	l2_multi
8	100	0,128916	1	HTTP	1	0	0	l2_learning
9	100	0,222519	2	HTTP	0	1	0	l2_pairs
10	100	0,019907	1	UDP	0	1	0	l2_pairs
11	100	0,009940	1	UDP	1	0	0	l2_learning
12	100	0,159536	1	HTTP	0	1	0	l2_pairs
13	100	0,436127	4	HTTP	0	0	1	l2_multi
14	100	0,105770	1	HTTP	1	0	0	l2_learning
15	100	0,089750	4	TCP	0	0	1	l2_multi

Tabela 14 - continuação da página anterior

	Qtde Pkt	Atraso	Topologia	Protocolo	Saídas Desejadas			
Amostra	**x1**	**x2**	**x3**	**x4**	**d1**	**d2**	**d3**	**C. Regras**
16	100	0,083463	3	TCP	0	0	1	l2_multi
17	100	0,003554	2	ICMP	0	1	0	l2_pairs
18	100	0,005420	4	ICMP	0	0	1	l2_multi
19	100	0,001492	1	ICMP	1	0	0	l2_learning
20	100	0,136936	1	HTTP	1	0	0	l2_learning
21	100	0,033548	2	UDP	0	1	0	l2_pairs
22	100	0,174505	1	HTTP	0	1	0	l2_pairs
23	100	0,048769	4	TCP	0	1	0	l2_pairs
24	100	0,249317	2	HTTP	1	0	0	l2_learning
25	100	0,458153	4	HTTP	0	1	0	l2_pairs
26	100	0,006427	4	ICMP	0	0	1	l2_multi
27	100	0,059285	2	TCP	0	1	0	l2_pairs
28	100	0,001783	1	ICMP	0	1	0	l2_pairs
29	100	0,004229	3	ICMP	1	0	0	l2_learning
30	100	0,002199	1	ICMP	0	0	1	l2_multi
31	100	0,064874	2	TCP	1	0	0	l2_learning
32	100	0,034307	2	UDP	0	1	0	l2_pairs
33	100	0,084436	3	TCP	0	0	1	l2_multi
34	100	0,036158	2	UDP	0	0	1	l2_multi
35	100	0,004208	3	ICMP	1	0	0	l2_learning
36	100	0,176837	1	HTTP	0	1	0	l2_pairs
37	100	0,050932	1	TCP	0	0	1	l2_multi
38	100	0,041514	1	TCP	1	0	0	l2_learning
39	100	0,046902	1	TCP	0	1	0	l2_pairs
40	100	0,001735	1	ICMP	0	1	0	l2_pairs
41	100	0,047670	3	UDP	1	0	0	l2_learning
42	100	0,106147	1	HTTP	1	0	0	l2_learning
43	100	0,058135	4	UDP	0	0	1	l2_multi
44	100	0,127456	1	HTTP	1	0	0	l2_learning
45	100	0,059188	2	TCP	0	1	0	l2_pairs
46	100	0,004728	3	ICMP	0	1	0	l2_pairs
47	100	0,355114	3	HTTP	1	0	0	l2_learning
48	100	0,285657	2	HTTP	0	0	1	l2_multi
49	100	0,237324	2	HTTP	0	1	0	l2_pairs
50	100	0,009327	1	UDP	1	0	0	l2_learning

Tabela 14 - continuação da página anterior

	Qtde Pkt	Atraso	Topologia	Protocolo	Saídas Desejadas			
Amostra	**x1**	**x2**	**x3**	**x4**	**d1**	**d2**	**d3**	**C. Regras**
51	100	0,056927	4	UDP	0	0	1	l2_multi
52	100	0,004461	3	ICMP	1	0	0	l2_learning
53	100	0,073546	3	TCP	0	1	0	l2_pairs
54	100	0,047738	3	UDP	1	0	0	l2_learning
55	100	0,022577	1	UDP	0	0	1	l2_multi
56	100	0,063558	2	TCP	1	0	0	l2_learning
57	100	0,001787	1	ICMP	0	1	0	l2_pairs
58	100	0,411827	4	HTTP	0	0	1	l2_multi
59	100	0,418913	4	HTTP	0	0	1	l2_multi
60	100	0,004874	3	ICMP	0	1	0	l2_pairs
61	100	0,030705	2	UDP	1	0	0	l2_learning
62	100	0,093385	4	TCP	0	1	0	l2_pairs
63	100	0,081120	4	TCP	0	0	1	l2_multi
64	100	0,229197	2	HTTP	1	0	0	l2_learning
65	100	0,155133	1	HTTP	0	1	0	l2_pairs
66	100	0,064349	2	TCP	1	0	0	l2_learning
67	100	0,107110	1	HTTP	1	0	0	l2_learning
68	100	0,003631	2	ICMP	0	0	1	l2_multi
69	100	0,051921	1	TCP	0	0	1	l2_multi
70	100	0,028599	2	UDP	1	0	0	l2_learning
71	100	0,001307	1	ICMP	1	0	0	l2_learning
72	100	0,014564	1	UDP	0	1	0	l2_pairs

A validação da rede, para os experimentos com o hardware-switch e com a quantidade de pacotes em 100, foi feita aplicando um conjunto de teste fornecido na Tabela 15, sobrescrevendo 18 das 36 amostras iniciais de teste e validação (30% das 120 amostras). Isso feito, separadamente, por experimentos com carga de pacotes (100, 1.000 e 10.000). Logo, para os experimentos com 1.000 pacotes foi utilizada tabela com 180 linhas e para os experimentos com 10.000 pacotes foi utilizada tabela com 1.800 linhas.

Tabela 15 – Conjunto de padrões de Validação - Experimentos com hardware-switch e com a quantidade de pacotes em 100

	Qtde Pkt	Atraso	Topologia	Protocolo	Saídas Desejadas			
Amostra	**x1**	**x2**	**x3**	**x4**	**d1**	**d2**	**d3**	**C. Regras**
1	100	0,280460	1	HTTP	1	0	0	l2_learning
2	100	0,004096	1	ICMP	1	0	0	l2_learning
3	100	0,151722	1	TCP	0	0	1	l2_multi
4	100	0,006379	1	ICMP	0	1	0	l2_pairs
5	100	0,350794	1	HTTP	0	0	1	l2_multi
6	100	0,009071	1	TCP	1	0	0	l2_learning
7	100	0,035979	1	UDP	0	1	0	l2_pairs
8	100	0,001881	1	ICMP	1	0	0	l2_learning
9	100	0,003278	1	ICMP	1	0	0	l2_learning
10	100	0,053361	1	UDP	0	0	1	l2_multi
11	100	0,006790	1	ICMP	0	1	0	l2_pairs
12	100	0,007084	1	ICMP	0	0	1	l2_multi
13	100	0,318858	1	HTTP	0	1	0	l2_pairs
14	100	0,065447	1	UDP	0	0	1	l2_multi
15	100	0,035074	1	UDP	0	1	0	l2_pairs
16	100	0,211026	1	HTTP	1	0	0	l2_learning
17	100	0,012675	1	UDP	1	0	0	l2_learning
18	100	0,341447	1	HTTP	0	0	1	l2_multi

A variação dos valores das entradas x1, x2, x3 e x4 (quantidade de pacotes, topologia, protocolo e atraso) serviram de base para a geração das saídas desejadas d1, d2 e d3 que juntas formam a classificação de um tipo conjunto de regra, a qual será mostrada no pós-processamento.

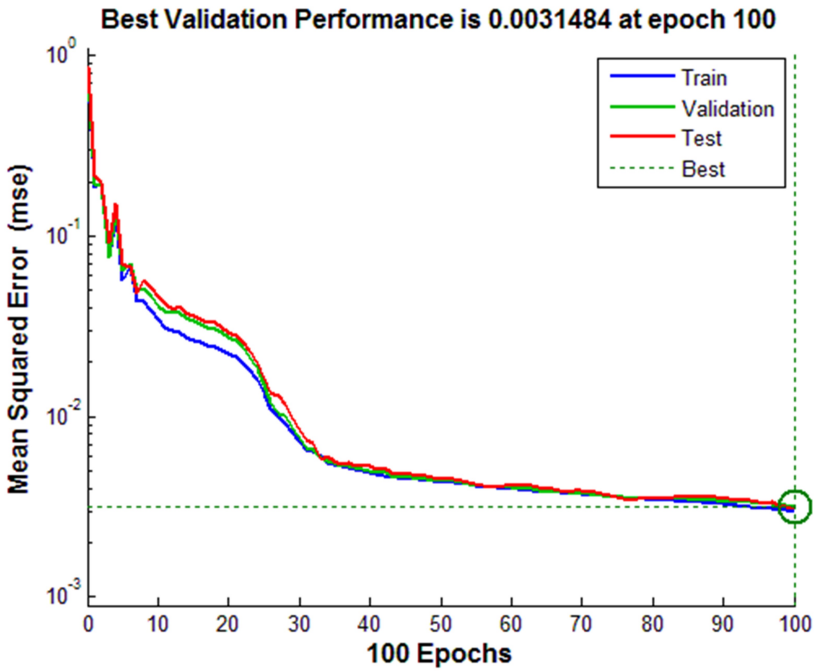
Dado que o problema se configura como um típico processo de classificação de padrões e que existe uma necessidade de adaptação das saídas que são números reais (entre 0 e 1) para as saídas representadas na Tabela 13. Foi necessário implementar uma rotina de pós-processamento das saídas fornecidas pela rede, mudando de valores reais para números inteiros. Para isso, foi adotado o critério de arredondamento a seguir, em que y_i representa as três saídas que foram arredondadas para 0 ou para 1.

$$y_i^{pós} = \begin{cases} 1, se\ y_i \geq 0{,}5 \\ 0, se\ y_i < 0{,}5 \end{cases}$$

As saídas pós-processadas da RNA com valores entre 0 e 0,49 foram arredondadas para 0. E as saídas com os valores iguais à 0,5 até 1 foram arredondadas para 1. A inicialização das matrizes de pesos foi feita com valores aleatórios apropriados. Utilizando as 480 amostras

iniciais como entradas, aplicando uma taxa de aprendizado de 0,1, uma precisão de 10^{-6} e utilizando a função de ativação logística (*logsig*) foi obtido, para o treinamento com 100 pacotes, o erro médio quadrático apresentado na Figura 30.

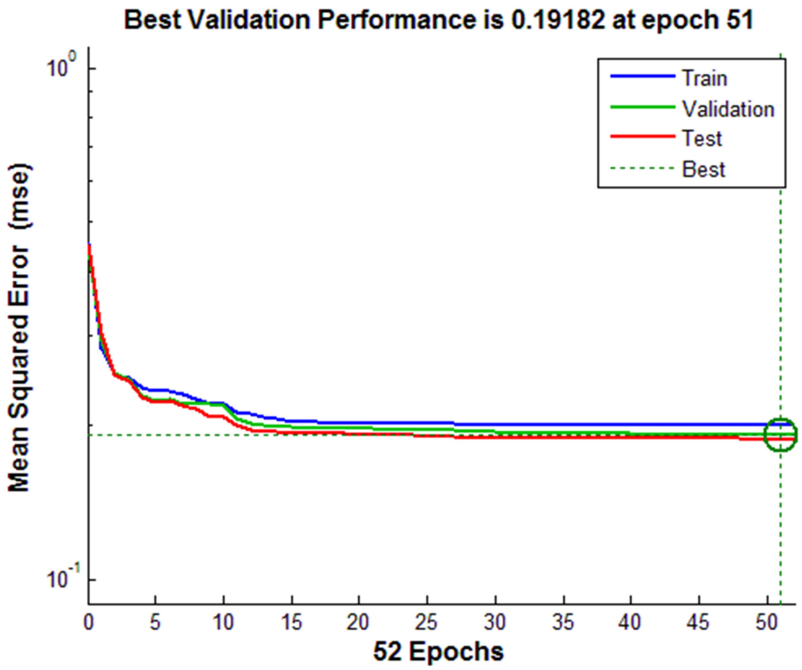
Figura 30 – Erro Médio Quadrático sem *momentum*



Fonte: (PRÓPRIO AUTOR)

Utilizando as mesmas 480 amostras iniciais como entradas, aplicando uma taxa de aprendizado de 0,1, uma precisão de 10^{-6}, utilizando a função de ativação logística (*logsig*) e fator de *momentum* de 0,9 foi obtido o gráfico de erro médio quadrático apresentado na Figura 31.

Figura 31 – Erro Médio Quadrático com *momentum*



Fonte: (PRÓPRIO AUTOR)

O treinamento utilizando o fator de *momentum* obteve a resposta num tempo mais rápido que no treinamento sem o fator de *momentum*. Como se pode observar através dos gráficos nas Figuras 30 e 31 , o segundo treinamento estabilizou o erro quadrático médio entre as épocas 10 e 15. O que não ocorreu com o treinamento anterior, no qual o erro quadrático médio veio caindo de forma gradual e lenta. O primeiro treinamento aconteceu em 2 segundos de tempo, sendo que o segundo não chegou a alcançar um quarto deste valor.

Os demais treinamentos seguiram os mesmos moldes, para o erro médio quadrático, dos apresentados para os experimentos com a quantidade de 100 pacotes.

4.2.3 Configuração das saídas desejadas

A saída deseja é um componente de validação utilizado no processo de treinamento de uma rede neural artificial. A seguir será explanado como foram imputados os valores desejados para a saída da RNA, os treinamentos e validações descritos anteriormente, foram fundamentados no ranqueamento gerado pelos resultados dos experimentos mostrados nesta seção.

Na Tabela 16 estão dispostos os intervalos de ranqueamento em milissegundos. O referido ranqueamento foi utilizado na configuração das saídas desejadas e, consequentemente, na validação da RNA com os experimentos demonstrados na Tabela 14. A topologia foi utilizada como fator de decisão no caso de algum valor de atraso coincidir. Desse modo, para os valores

demonstrados, os atrasos entre 0,04 e 0,07 milissegundos podem abarcar tanto o protocolo UDP quanto o protocolo TCP, porém os mesmos se diferem quando a topologia é comparada. O valor 0,041 milissegundos, por exemplo, se estiver na topologia 1 é oriundo de um experimento com o protocolo TCP e a sugestão de regra da RNA é a *l2-learning*. Já se estiver com a topologia 3 foi testado com o protocolo UDP e a sugestão de regra da RNA é a *l2-multi*.

Tabela 16 – Intervalos de ranqueamento utilizados na configuração das saídas desejadas - Experimentos com o Mininet e com a quantidade de pacotes em 100

Intervalos de ranqueamento (em ms)			
Protocolo	**learning**	**pairs**	**multi**
ICMP - 0,001 e 0,007	0,0010 - 0,0015	0,0016 - 0,0020	0,0021 - 0,0025
	0,0026 - 0,0030	0,0031 - 0,0035	0,0036 - 0,0040
	0,0041 - 0,0045	0,0046 - 0,0050	0,0051 - 0,0055
	0,0056 - 0,0060	0,0061 - 0,0065	0,0066 - 0,0070
UDP - 0,009 e 0,07	0,009 - 0,010	0,011 - 0,020	0,021 - 0,025
	0,026 - 0,030	0,031 - 0,035	0,036 - 0,040
	0,046 - 0,050	0,051 - 0,055	0,041 - 0,045
	0,061 - 0,065	0,066 - 0,070	0,056 - 0,060
TCP - 0,04 e 0,1	0,040 - 0,045	0,046 - 0,050	0,051 - 0,055
	0,061 - 0,065	0,056 - 0,060	0,066 - 0,070
	0,076 - 0,080	0,071 - 0,075	0,081 - 0,085
	0,096 - 0,100	0,091 - 0,095	0,086 - 0,090
HTTP - 0,1 e 0,5	0,10 - 0,13	0,15 - 0,17	0,19 - 0,20
	0,25 - 0,26	0,20 - 0,23	0,29 - 0,30
	0,35 - 0,36	0,30 - 0,33	0,38 - 0,39
	0,49 - 0,50	0,46 - 0,47	0,40 - 0,42

Na Tabela 17 estão dispostos os intervalos de ranqueamento em milissegundos. O referido ranqueamento foi utilizado na configuração das saídas desejadas e, consequentemente, na validação da RNA com os experimentos demonstrados na Tabela 15.

Tabela 17 – Intervalos de ranqueamento utilizados na configuração das saídas desejadas - Experimentos com o hardware-switch e com a quantidade de pacotes em 100

Intervalos de ranqueamento (em ms)			
Protocolo	**learning**	**pairs**	**multi**
ICMP - 0,003 e 0,009	0,0050 - 0,00599	0,0060 - 0,00699	0,0070 - 0,00799
UDP - 0,01 e 0,05	0,01 - 0,02	0,03 - 0,0399	0,04 - 0,0499
TCP - 0,09 e 0,2	0,09 - 0,099	0,1 - 0,12	0,155 - 0,165
HTTP - 0,2 e 0,4	0,258 - 0,268	0,3 - 0,31	0,36 - 0,365

4.2.4 Acertos e erros obtidos com a RNA

Realizando a validação da rede, obteve-se a matriz de confusão representada na Figura 32, com os percentuais de acerto em cada fase: treinamento, validação e teste da rede neural artificial.

Figura 32 – Matriz de Confusão de Acertos e Erros

Fonte: (PRÓPRIO AUTOR)

A matriz gerada esta divida em 4 matrizes: uma de treinamento, uma de validação, uma de teste a uma que junta os 3 valores anteriores formando uma matriz de resultados geral. Todas as 4 matrizes mostram um comparativo das saídas da RNA com as saídas desejadas pontuando os acertos e erros no formato de percentual. Avaliando os quadros de treinamento, teste e validação, pode-se perceber que os percentuais de acertos e erros se mantiveram com uma pequena taxa de variação. A taxa de percentual de acertos acima de 90% confirma a eficiência da rede treinada para a aplicação.

A velocidade de treinamento variou de acordo a presença e ausência do fator de *momentum* e da quantidade de camadas escondidas. A velocidade de resposta e a quantidade de camadas escondida, para esse tipo de problema, não foram fatores preocupantes. No caso da velocidade de resposta quanto menor era seu tempo, maior era o fator de *momentum* e mais rápido era o treinamento. Os valores de erro médio quadráticos (com e sem *momentum*) e a geração da matriz de confusão, foram resultados da melhor rede treinada entre 10 repetições.

4.3 Comparação entre o modelo de inserção de regras tradicional e o modelo gerado pela RNA

A integração da SDN com o método de decisão, respaldada em inteligência computacional, proporcionou novas abstrações para as formas de encaminhamento e programação da rede, desse modo a intenção é exibir os resultados comparativos entre o modelo de inserção de regras tradicional e o modelo gerado pela RNA para que algumas conclusões possam ser tomadas no que diz respeito a utilização da ferramenta.

Todos os dados de comparação foram feitos da seguinte maneira, pegou-se todos os valores gerados por cada uma das regras dinâmicas individualmente, depois os valores gerados pela RNA e confrontou-se um com o outro.

A Tabela 18 lista a média dos intervalos de confiança para os atrasos, em milissegundos, de todas as regras rodando individualmente, além da média da resposta obtida caso seja feita a opção sugerida pela RNA. Isso para a *l2-learning.py*, para a *l2-pairs.py* e para a *l2-multi.py* associadas à regra estática com conjunto de *match* por porta.

Tabela 18 – Comparação do modelo de inserção de regras tradicional e do modelo gerado pela RNA - *match* por porta

Qtde Pkt	**learning**	**pairs**	**multi**	**resposta da RNA**
100	0,103125 - 0,104256	0,100181 - 0,101212	0,109095 - 0,110174	0,090964 - 0,091914
1000	0,173731 - 0,174763	0,169184 - 0,170221	0,171235 - 0,172205	0,149854 - 0,150813
10000	0,330563 - 0,331382	0,336444 - 0,337294	0,331817 - 0,332627	0,305856 - 0,306668

A Tabela 19 lista a média dos intervalos de confiança para os atrasos, em milissegundos, de todas as regras rodando individualmente, além da média da resposta obtida caso seja feita a opção sugerida pela RNA. Isso para a *l2-learning.py*, para a *l2-pairs.py* e para a *l2-multi.py* associadas à regra estática com conjunto de *match* por MAC.

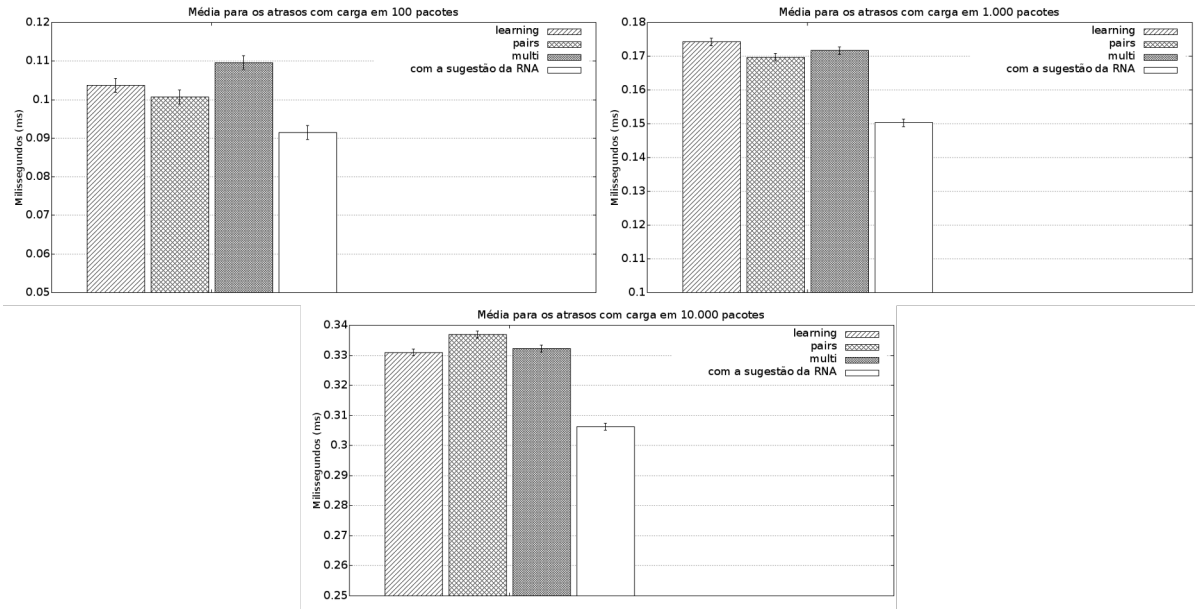
Tabela 19 – Comparação do modelo de inserção de regras tradicional e do modelo gerado pela RNA - *match* por MAC

Qtde Pkt	learning	pairs	multi	resposta da RNA
100	0,148952 - 0,150420	0,139066 - 0,140397	0,152591 - 0,154008	0,128271 - 0,129524
1000	0,199864 - 0,201031	0,203737 - 0,204941	0,205057 - 0,206228	0,184905 - 0,186058
10000	0,366636 - 0,367445	0,365900 - 0,366748	0,379105 - 0,379924	0,354539 - 0,355347

Os valores apresentados a seguir são um comparativo do desempenho dos três conjuntos de regras do POX com a resposta da RNA. Com a quantidade de pacotes em 100, variando a topologia e os protocolos. Os resultados foram utilizados para o treinamento e validação da RNA. Associadas à regra estática com conjunto de *match* por porta e também à regra estática com conjunto de *match* por MAC.

A Figura 33 mostra um comparativo entre o modelo de inserção de regras tradicional e o modelo sugerido pela RNA, fixados na média e nos intervalos de confiança para os atrasos em milissegundos. Para a regra estática de *match* por porta e com as 3 cargas de pacotes (100, 1.000 e 10.000).

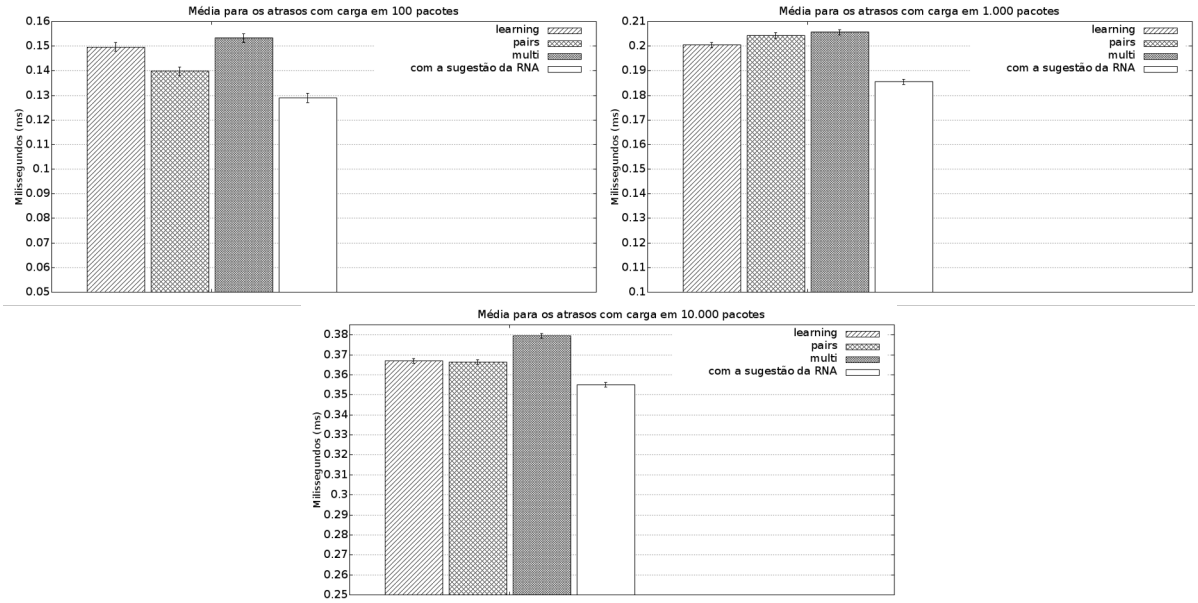
Figura 33 – Comparativo entre o modelo de inserção de regras tradicional e o modelo da RNA - *match* por porta



Fonte: (PRÓPRIO AUTOR)

E por fim, na Figura 34 é possível observar um comparativo entre o modelo de inserção de regras tradicional e o modelo sugerido pela RNA, baseados na média e nos intervalos de confiança para os atrasos em milissegundos, para a regra estática com conjunto de *match* por MAC e com as 3 cargas de pacotes (100, 1.000 e 10.000).

Figura 34 – Comparativo entre o modelo de inserção de regras tradicional e o modelo da RNA - *match* por MAC



Fonte: (PRÓPRIO AUTOR)

A resposta da RNA obteve o melhor desempenho se comparado a todas as outras possibilidades utilizando o modelo tradicional de inserção de regras, ou seja, fixando apenas um tipo de regra dinâmica de encaminhamento de pacotes. Para os testes com o hardware-switch observou-se o mesmo comportamento do que o demonstrado com o virtualizador Mininet. Porém os valores de atraso médio gerados, como explanado no comparativo entre os 2 testes, são mais altos (tanto para - *match* por porta quanto para - *match* por MAC).

A sugestão de qual regra deve ser utilizada pelo controlador dependerá do estado em que a SDN se encontrar, ou seja, para cada combinação de topologia, com protocolo e carga de pacotes uma das 3 regras dinâmicas obteve um resultado de performance, em termos de atraso médio, diferente e em cada um deles uma regra é melhor do que a outra. Se a regra for utilizada de forma correta o ganho em desempenho cresce significativamente como mostraram os resultados.

4.3.1 Custo operacional gerado com a troca de regras no controlador

Mesmo conhecendo determinadas funções e características de cada regra (e parecer óbvio a tomada de decisão) a utilização da inteligência computacional serve para classificação automática em um ambiente com uma variação de topologias, protocolos e cargas de pacotes.

Logo, a sugestão de classificação automática serve para facilitar a operação mantendo sempre um nível de excelência da prática operacional.

O custo de alterar o conjunto de regras no controlador foi descoberto, por métodos empíricos realizados através de experimentos em laboratório, é algo com valores na fração de segundos, com aproximadamente 0,5 segundo para descer a regra ativa e com aproximadamente mais 0,5 segundo para subir a nova regra. Contudo, entre uma troca de regras, o switch muda automaticamente sua configuração de atuação para modo *legacy* (caso encontre alguma inatividade da rede).

Com base nos resultados apresentados na subseção anterior, mantendo a mesma topologia, protocolo e carga de pacotes por um tempo e optando pela regra com a melhor performance, houve um ganho significativo quando comparado à utilização de uma regra inferior para a mesma configuração da rede. Já o custo operacional gerado com a troca de regras do controlador irá depender da quantidade de vezes que for necessária a troca e de quanto tempo a rede permanecer em operação. De modo geral a classificação sendo inserida como sugestão para o operador foi considerada a melhor solução para o método de inteligência computacional testado.

Dentro do contexto deste trabalho, o único problema seria a alteração de regras em uma rede em produção que possua características de alteração de protocolos e de topologia a todo tempo, pois alteração de carga de pacotes já é algo praticável em quase todo tipo de rede. Neste perfil de rede o custo operacional gerado com a troca de regras subiria ocasionando uma perda de performance, porém não seria algo que preocuparia pois no formato como está sendo feito a classificação de regras, como uma sugestão para o operador que pode observar as características da rede que está supervisionando, não causa nenhum tipo de problema.

5 CONCLUSÕES E TRABALHOS FUTUROS

As SDNs e o protocolo OpenFlow são temas atuais e o ramo pesquisa, a eles associado, encontra-se num momento de alto crescimento, além de grandes marcas de switches estarem investindo no segmento.

A realização de testes respaldados em cenários variados rendem bons frutos no auxílio em áreas de pesquisa e a redes corporativas inseridas nesse contexto. Além da inserção de um mecanismo inteligente, utilizando RNA, que classifique as regras para o controlador SDN e que mantenha os requisitos predeterminados para a rede, dentro dos parâmetros de QoS, ser algo muito interessante dentro desse ramo de pesquisa.

A proposta do trabalho foi fornecer, através do auxílio operacional de redes SDN utilizando inteligência computacional, um aumento da performance dos switches OpenFlow e fornecer subsídio para pesquisadores e profissionais de redes trabalharem e evoluírem no ramo. Além de medir o desempenho dos comutadores nos diversos tópicos gerados pelos testes e experimentos. A pesquisa ficou restrita a regras dinâmicas associadas a apenas um tipo de regra estática e a resposta gerada por um treinamento supervisionado da RNA impossibilita a atualização em tempo real.

Os resultados mostraram que a utilização de um método de tomada de decisão fornece um ganho significativo de desempenho para redes SDN de mesmo perfil. Perfis inseridos nas configurações e testes apresentados nesse trabalho. Ademais, os experimentos contribuíram com valores que refletiram um ganho de performance e investir nesse ramo de pesquisa contribuirá na operação e no cotidiano do profissional de redes nesse segmento. A utilização do hardware-switch, de forma geral, seguiu os mesmos moldes do simulador Mininet, deixando a decisão de utilização de um equipamento físico equiparada à utilização de um switch virtual (que são bastante utilizados no ramo de redes definidas por software).

O custo de alterar o conjunto de regras no controlador foi descoberto e com base, também, nos resultados apresentados, conclui-se que mantendo a mesma topologia, protocolo e carga de pacotes por um tempo e optando pela regra com a melhor performance, haverá um ganho significativo quando comparado à utilização de uma regra inferior para a mesma configuração da rede. Já o custo operacional gerado com a troca de regras do controlador irá depender da quantidade de vezes que for necessária a troca e de quanto tempo a rede permanecer em operação. De maneira geral a classificação sendo inserida como sugestão para o operador foi

considerada a melhor solução para o método de inteligência computacional testado, resultado influenciado pelo método supervisionado de treinamento da RNA.

Dentro do contexto deste trabalho, o único problema seria a alteração de regras em uma rede em produção que possua características de alteração de protocolos e de topologia a todo tempo, pois alteração de carga de pacotes já é algo praticável em quase todo tipo de rede. Neste perfil de rede o custo operacional gerado com a troca de regras subiria ocasionando uma perda de performance, porém não seria algo que preocuparia pois no formato como está sendo feito a classificação de regras, como uma sugestão para o operador que pode observar as características da rede que está supervisionando, não causa nenhum tipo de problema.

Como trabalhos futuros serão testados outras aplicações, oriundas de outros protocolos, também serão testadas regras aplicadas em outras camadas com funcionalidades em nível de rede e transporte, por exemplo. A utilização de hardware-switches e a aplicação da classificação inteligente, a quente, em uma rede em produção. Serão testados outros tipos de algoritmos /métodos de inteligência artificial. E, por fim, também, serão testados outra versão do protocolo OpenFlow e outro tipo de controlador SDN.

REFERÊNCIAS

ALVES, A. R. et al. Homenetrescue: An sdn service for troubleshooting home networks. In: IEEE. **NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium**. [S.l.], 2018. p. 1–7.

BISHOP, C. M. **Neural networks for pattern recognition**. 5th. ed. [S.l.]: Oxford university press, 1995.

CHEN, Y. et al. Adaptive distributed software defined networking. **Computer Communications**, Elsevier, v. 102, p. 120–129, 2017.

COSTA, L. C. et al. **Balanceamento de carga utilizando planos de dados OpenFlow comerciais**. Dissertação (Mestrado) — Universidade Federal de Juiz de Fora, 2016.

COSTA, L. R. **Openflow e o paradigma de redes definidas por software**. Dissertação (Mestrado) — Universidade de Brasília, 2013. Disponível em: <http://bdm.unb.br/handle/10483/5674>.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The road to sdn: an intellectual history of programmable networks. In: ACM. **ACM SIGCOMM Computer Communication Review**. [S.l.], 2014. v. 44, n. 2, p. 87–98.

GOMES, V. S. et al. Flowvisorqos: Aperfeicoando o flowvisor para aprovisionamento e recursos em redes virtuais definidas por software. In: **IV Workshop de Pesquisa Experimental a Internet do Futuro-Simposio Brasileiro de Redes de Computadores e Sistemas Distribuidos (SBRC 2013), Brasilia**. [S.l.: s.n.], 2013. p. 35–41.

GUDE, N. et al. Nox: towards an operating system for networks. In: ACM. **ACM SIGCOMM Computer Communication Review**. [S.l.], 2008. v. 38, n. 3, p. 105–110.

GUEDES, D. et al. Redes definidas por software: uma abordagem sistêmica para o desenvolvimento de pesquisas em redes de computadores. **Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC 2012**, v. 30, n. 4, p. 160–210, 2012.

HAGAN, M. T. et al. **Neural network design**. [S.l.]: Pws Pub. Boston, 1996. v. 20.

HAYKIN, S.; NETWORK, N. A comprehensive foundation. **Neural networks**, IEEE, v. 2, n. 204, p. 41, 2004.

JUNIPER, N. **Decodificando a SDN**. 2017. Http://www.juniper.net/br/pt/dm/sdn-wp/. Acesso em: 12/2018.

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. **Proceedings of the IEEE**, IEEE, v. 103, n. 1, p. 14–76, 2015.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. In: ACM. **Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks**. [S.l.], 2010. p. 19.

MACAPUNA, C. A. B. et al. **Proposta e validação de nova arquitetura de redes de data center**. Dissertação (Mestrado) — UNICAMP – Universidade Estadual de Campinas, Campinas - SP, Abril 2011.

MASTERS, T. **Practical neural network recipes in C++**. [S.l.]: Morgan Kaufmann, 1993.

MATLAB, R. **The Matlab - High-performance Interactive Software for Numerical Calculation**. 2018. Https://www.mathworks.com/. Acesso em: 12/2018.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943.

MCKEOWN, N. et al. Openflow: enabling innovation in campus networks. **ACM SIGCOMM Computer Communication Review**, ACM, v. 38, n. 2, p. 69–74, 2008.

MESTRES, A. et al. Knowledge-defined networking. **ACM SIGCOMM Computer Communication Review**, ACM, v. 47, n. 3, p. 2–10, 2017.

MININET, C. **Mininet An Instant Virtual Network on your Laptop (or other PC)**. 2018. Http://www.mininet.org/. Acesso em: 02/2019.

NASCIMENTO, M. R. et al. Routeflow: Roteamento commodity sobre redes programáveis. In: SBRC. **XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos-SBRC**. [S.l.], 2011. p. 44–58.

NOX, R. **The NOX Controller - NOX Repo**. 2018. Http://www.noxrepo.org/nox/. Acesso em: 03/2018.

OSTINATO, R. **The Ostinato Packet Generator - OSTINATO Repo**. 2018. Https://ostinato.org/. Acesso em: 12/2018.

OVS, C. **Production Quality, Multilayer Open Virtual Switch**. 2018. Http://www.openvswitch.org/. Acesso em: 09/2018.

PFAFF, B. et al. Extending networking into the virtualization layer. In: **Hotnets**. [S.l.: s.n.], 2009. v. 15, p. 24–38.

POX, R. **The POX Controller - NOX Repo**. 2018. Https://github.com/noxrepo/pox/. Acesso em: 09/2018.

QAZI, Z. A. et al. Application-awareness in sdn. In: ACM. **ACM SIGCOMM computer communication review**. [S.l.], 2013. v. 43, n. 4, p. 487–488.

RAUBER, T. W. Redes neurais artificiais. **Universidade Federal do Espírito Santo**, 2005.

RUCKERT, J. et al. Demo: Software-defined network service chaining. In: **Third European Workshop on Software Defined Networks**. [S.l.: s.n.], 2014. p. 139–40.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, Nature Publishing Group, v. 323, n. 6088, p. 533, 1986.

SABBEH, A. et al. Performance prediction of software defined network using an artificial neural network. In: IEEE. **SAI Computing Conference (SAI), 2016**. [S.l.], 2016. p. 80–84.

SHERWOOD, R. et al. Flowvisor: A network virtualization layer. **OpenFlow Switch Consortium, Tech. Rep**, v. 1, p. 132, 2009.

SHIN, M.-K.; NAM, K.-H.; KIM, H.-J. Software-defined networking (sdn): A reference architecture and open apis. In: IEEE. **ICT Convergence (ICTC), 2012 International Conference on**. [S.l.], 2012. p. 360–361.

SILVA, I. d.; SPATTI, D. H.; FLAUZINO, R. A. Redes neurais artificiais para engenharia e ciências aplicadas. In: ARTLIBER. [S.l.], 2010. p. 33–111.

STANLEY, K. O.; MIIKKULAINEN, R. Evolving neural networks through augmenting topologies. **Evolutionary computation**, MIT Press, v. 10, n. 2, p. 99–127, 2002.

SUJITHA, S.; MANIKANDAN, M.; ASHWINI, G. Sdn controller. In: **Innovations in Software-Defined Networking and Network Functions Virtualization**. [S.l.]: IGI Global, 2018. p. 72–99.