CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

Lucas Alcântara de Souza

PROPOSTA DE IMPLEMENTAÇÃO DE UM FIREWALL PARA UMA INSTITUIÇÃO DE ENSINO BASEADO EM REDES DEFINIDAS POR SOFTWARE

Timóteo

2019

Lucas Alcântara de Souza

PROPOSTA DE IMPLEMENTAÇÃO DE UM FIREWALL PARA UMA INSTITUIÇÃO DE ENSINO BASEADO EM REDES DEFINIDAS POR SOFTWARE

Monografia apresentada à Coordenação de Engenharia de Computação do Campus Timóteo do Centro Federal de Educação Tecnológica de Minas Gerais para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Me. Talles Quintão Pessoa

Timóteo

2019

Lucas Alcântara de Souza

Proposta de implementação de um Firewall baseado em redes definidas por software

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais, campus Timóteo, como requisito parcial para obtenção do título de Engenheiro de Computação.

Trabalho aprovado. Timóteo, 08 de Agosto de 2019:

Prof. Me. Talles Quintão Pessoa
Orientador

Prof. Me. Adilson Mendes Ricardo
Professor Convidado

Prof. Me. Gustavo Henrique dos Santos Ribeiro
Professor Convidado

Timóteo
2019

.

.

.

Dedico a
Deus.

Agradecimentos

Agradeço primeiramente a Deus por ter me dado a força e persistência necessária para a conclusão dessa etapa na minha vida.

Agradeço aos meus pais Romildo Trindade de Souza e Eliene alcântara da Silva Souza pela paciência, pelo apoio, pelos incentivos durante todo esse tempo de graduação.

Agradeço ao meu orientador Talles Quintão Pessoa pela paciência, pela disponibilidade e pelo auxílio na realização deste trabalho.

Agradeço aos meus amigos Débora, Gustavo, Helton, Jefferson e Rafael que estiveram presentes e atuantes nesta caminhada e principalmente um agradecimento especial a Bruna Cristina.

Um agradecimento a todos os professores que contribuíram para minha formação e por fim, a todos que de alguma forma contribuíram direta ou indiretamente para a conclusão desta graduação e desse trabalho, o meu muito obrigado!

“Os grandes navegadores
devem sua reputação aos temporais e tempestades”.
Epicuro

Resumo

O crescente uso das redes de computadores tem representado a solução para muitos problemas da atualidade, a Internet como conhecemos hoje é basicamente a interconexão de milhares de redes de computadores espalhadas pelo mundo compartilhando mensagens e dados por um protocolo comum. Tal grandiosidade representa sua maior limitação, que acabou tornando o gerenciamento de redes baseadas no protocolo IP complexas e difíceis. O paradigma de redes definidas por softwares surgiu como uma possibilidade de resolver o problema de gerenciamento das redes de computadores com uma proposta de separação do plano de controle do plano de dados dos dispositivos de encaminhamento da rede. Tal proposta permite que os operadores de rede tenham uma visão completa da rede a partir de um controlador de rede, assim eliminando o problema de ter que configurar cada dispositivo de rede separadamente. A segurança nas rede de computadores vai muito além das informações que trafegam pela mesma, sendo necessária a garantia e a confiabilidade dos dados. A adoção do paradigma de Redes Definidas por Software (SDN) implica na utilização de aplicações de rede, que basicamente são serviços executados no controlador da rede, sendo o controlador de rede o software programável onde se configura as regras da rede, consequentemente o uso de dispositivos de segurança legados como o *firewall* precisam ser reprojetados e implementados como aplicação na rede. Sendo assim, este trabalho de pesquisa tem como objetivo formular uma proposta do uso de *firewall* utilizando o paradigma SDN e simular a sua aplicação na rede local de uma instituição de ensino como um cenário de estudo de caso. As simulações demonstraram a viabilidade e as limitações da utilização do paradigma SDN em redes locais.

Palavras-chave: *firewall*; redes definidas por software; segurança de rede.

Abstract

The growing use of computer networks has represented the solution to many problems today, the Internet as we know it today is basically the interconnection of thousands of computer networks scattered around the world sharing messages and data by a common protocol. Such greatness represents its greatest limitation, which has made the management of networks based on IP protocol complex and difficult. The paradigm of software-defined networks emerged as a possibility to solve the problem of management of computer networks with a proposal to separate the control plan of the data plan of the routing devices of the network. Such a proposal allows network operators to have a complete view of the network from a network controller, thus eliminating the problem of having to configure each network device separately. The security in the network of computers goes much beyond the information that travels by the same, being necessary the guarantee and the reliability of the data. The adoption of the *Software Define Network* (SDN) paradigm implies the use of network applications, which are basically services executed in the network controller, the network controller being the programmable software where the network rules are configured, consequently the use of Legacy security devices like the firewall need to be redesigned and deployed as an application on the network. Thus, this research aims to formulate a proposal of the use of firewalls using the sdn paradigm and to simulate its application in the local network of a teaching institution as a scenario of case study. The simulations demonstrated the feasibility and limitations of using the SDN paradigm in local networks.

Keywords: firewall; software define network; network security.

Lista de ilustrações

Lista de tabelas

Lista de códigos

Lista de abreviaturas e siglas

API	*Application Programming Interface*
ARP	*Address Resolution Protocol*
Darpa	*Defense Advanced Research Projects Agency*
DMZ	*Demilitarized Zone*
DNS	*Domain Name System*
FTP	*File Transfer Protocol*
ICMP	*Internet Control Message Protocol*
IDS	*Intrusion detection System*
IETF	*Internet Engineering Task Force*
IoT	*Internet of Things*
IP	*Internet protocol*
IPS	*Intrusion Prevention System*
ISP	*Internet Service Provider*
LAN	*Local Area Network*
MAC	*Media Access Control*
NBI	*Northbound interface*
NIC	*Network Interface Card*
NOS	*Network Operating System*
P2P	*Peer-to-peer*
RTT	*Round-Trip time*
SBI	*Southbound interface*
SDN	*Software Define Network*
SSH	*Secure Shell*
SSL	*Secure Socket Layer*
STP	*Spanning Tree Protocol*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*

Sumário

1 Introdução

Atualmente, as redes de computadores possuem limitações significativas relacionadas ao gerenciamento, a flexibilidade e a adaptabilidade a novas aplicações. O modelo das redes tradicionais, baseado no *Internet protocol* (IP), são complexas e difíceis de gerenciar, pois para a aplicação de políticas e configuração dos dispositivos os operadores de rede precisam configurar cada dispositivo de rede separadamente usando comandos de baixo nível e geralmente específicos de empresas proprietárias (MASOUDI; GHAFFARI, 2016).

As redes de computadores são constituídos por diversos dispositivos, como roteadores e *switches*, que possuem mecanismos de controle e encaminhamento de dados. Os planos de dados e os planos de controle desses dispositivos são fortemente entrelaçados, ou seja, são agrupados dentro dos dispositivos de redes. (KREUTZ et al., 2015).

Os dispositivos de redes possuem fornecedores distintos o que implica em propriedades também distintas e a implantação de dispositivos diferentes em uma rede pode incindir em problema de interfaces incompatíveis entre os dispositivos, sendo assim a integração vertical dos dispositivos bem como a variedade de fornecedores dificultam a inovação e o avanço na área das redes de computadores e implementação de medidas de segurança (BOUCADAIR; JACQUENET, 2014).

Os *firewalls* são dispositivos voltados para a segurança de redes, sua função é filtrar o tráfego de entrada ou saída de redes privadas, ou seja, servindo como uma barreira de segurança (GUPTA; KAUR; KAUR, 2016). No modelo tradicional de redes, *firewalls* convencionais supõem de que usuários internos de uma rede privada são confiáveis e qualquer usuário externo é um possível inimigo para a rede (BELLOVIN, 1999). Acontece que essa suposição já não é valida há muito tempo, pois usuários internos poderiam lançar ataques a outros usuários facilmente driblando o *firewall* implementado na rede (SCHULTZ, 2002).

Uma alternativa para a solução desse problema é um cenário em que é proposto uma implementação de regras para a manipulação do plano de dados como módulos de software em vez de mantê-los acoplados ao hardware. Essa proposta permitiria aos administradores de rede ter um controle maior sobre o tráfego da rede e portanto oferecer um grande ganho de desempenho em termos do uso eficiente dos recursos da rede (HU; HAO; BAO, 2014). Portanto utilizando do paradigma de *Software Define Network* (SDN), este trabalho visa a elaborar uma proposta de um sistema na definição de regras de um *firewall* para a aplicação de redes locais para uma instituição de ensino.

1.1 Problema

As infraestruturas de redes existente são altamente complexas sendo baseadas em hardware, difíceis de gerenciar e de implementar regras de segurança. Roteadores, *Switches* e *firewalls* são diferentes dispositivos que constituem uma rede de computadores e cada um

desses dispositivos executam softwares que geralmente são fechados e específicos de determinados fornecedores. Para o gerenciamento e definições de regras de segurança de um ambiente contendo muitos desses dispositivos são necessárias configurações individuais conforme a especificação do seu fornecedor, tornando-se uma tarefa bastante árdua para os administradores de redes (KAUR et al., 2015).

Nesse cenário de inúmeros dispositivos, a segurança da informação é fundamental em um mundo altamente dependente do uso das tecnologias da informação. As vulnerabilidades de seguranças em uma rede acadêmica, possibilita que dados vitais possam ser hackeados ou ter o desempenho prejudicado por uma pessoa mal-intencionada.

O *firewall* é a maneira mais comum e eficiente para a proteção e controle do tráfego de dados em uma rede local. Apesar de suas vantagens, os *firewalls* tradicionais apresentam algumas desvantagens entre elas o custo elevado das soluções proprietárias (SATASIYA et al., 2016).

O paradigma SDN é uma alternativa de como se projeta e gerencia redes, pois propõem uma separação do plano de controle do plano de dados, facilitando o gerenciamento da rede e permitindo a implementação de novas aplicações e o gerenciamento simplificado de protocolos (SATASIYA et al., 2016).

1.2 Objetivos

Propor um sistema de definição de regras de um *firewall* utilizando *Software Define Network* para a aplicação em uma rede local de uma instituição de ensino.

1.2.1 Objetivos Específicos

Além do principal objetivo proposto neste trabalho, será apresentado os seguintes objetivos específicos:

1. Levantar dos tipos de controladores utilizados em Redes Definidas por Software;
2. Definir as regras do *firewall* baseado em SDN;
3. Simular o ambiente proposto para validação das regras propostas.

1.3 Estrutura do texto

Este trabalho está organizado em capítulos: o capítulo 2 traz o referencial teórico sobre o tema, fornecendo uma discussão detalhada sobre os principais itens da literatura a respeito do tema do trabalho. O Capítulo 3 apresenta o estado da arte relacionado ao tema de pesquisa. O Capítulo 4 traz a relação dos materiais de desenvolvimento e explicações a respeito dos métodos utilizados para realização do trabalho. O Capítulo 5 apresenta a forma como foi preparado o ambiente de simulação. O Capítulo 6 expõe os resultados obtidos a partir da experimentação do objetivo e algumas discussões a respeito do processo. Por fim, o Capítulo 7 concluí e indica propostas de trabalhos futuros.

2 Fundamentos teóricos

Este capítulo foi dividido em três partes: a seção 2.1 apresenta uma fundamentação teórica sobre as redes IP, listando as camadas redes e suas definições. A Seção 2.2 caracteriza os diferentes tipos de *firewall* e finalmente a seção 2.3 apresenta os conceitos sobre o paradigma SDN.

2.1 Redes e IP

O computador é tido como uma das grandes conquistas tecnológicas alcançadas pela humanidade durante o seculo XX, desde sua invenção onde apresentava grandes estaturas até o surgimento dos computadores pessoais e a adoção ao grande público (TANENBAUM, 2011). Acontece que individualmente o computador não agrega muito valor as comunicações, foi quando então houve a ideia de combinar o uso do computador às comunicações, surgindo então a ideia de juntar grupos de computadores para a troca de informações entre si.

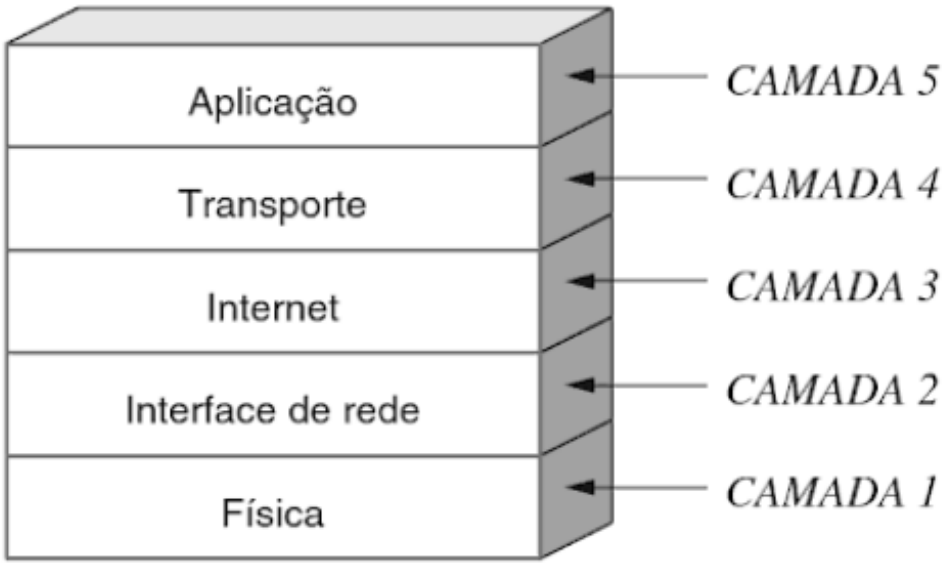
A capacidade de troca de informações entre um grupo de computadores é conhecido como uma rede, a rede não é necessariamente composta somente por computadores, mas por quaisquer dispositivos que tenham a capacidade de troca de informações como o computador (FOROUZAN; FEGAN, 2009).

Tendo em vista a formação de uma rede e então pensando na interligação dessa rede com outras redes similares ou não, deu inicio ao conceito de internet. Atualmente a Internet como é conhecida, é composta por um ecossistema composto pela existência de não só uma ou duas, mas de milhares de redes e seus dispositivos conectadas e comunicando entre si de forma a alcançar todo o mundo moderno e atendendo a necessidade mundial, facilitando a troca de informações e aproximação de pessoas (FOROUZAN; FEGAN, 2009).

Pensando na dimensão alcançada pela Internet é natural pensar em como todo esse ambiente se mantém coeso e funcionando. A tecnologia por trás desse fenômeno foi inicialmente criada nos Estados Unidos durante os anos 70 pela Darpa, pensando justamente nessa interligação de redes, foi então quando nasceu a ideia de interligar as redes através de uma estratégia em camadas, onde cada camada realizaria uma função específica na atividade envolvida na camada finalizando por passar adiante para as outras camadas realizarem suas funções próprias (TANENBAUM, 2011).

Como foi dito por Tanenbaum (2011), a tecnologia por trás do bom funcionamento da Internet mundial funciona por camadas. (COMER, 2016b) termina explicando que essa estratégia em camadas tem como objetivo a divisão dos problemas envolvidos na interligação de redes em tarefas que pudessem ser realizadas por protocolos, onde cada camada envolvida no processo de interligação teria seus próprios protocolos funcionando de forma harmoniosa resultando em um processo que dois protocolos de camadas distintas de forma alguma realizem a mesma função.

Figura 1 – Modelo em camadas TCP/IP.



Fonte: (COMER, 2016b)

Por fim, essa visão da divisão de tarefas envolvidas na interligação de redes em protocolos ficou conhecida como modelo de interligação por TCP/IP. A figura 1 ilustra a arquitetura de protocolos por trás da Internet de acordo com (COMER, 2016b) e (TANENBAUM, 2011).

As próximas seções descrevem resumidamente a função e protocolos envolvidos no funcionamento da arquitetura TCP/IP apresentado pela figura 1.

2.1.1 Camada 1: Física

Um dos princípios básicos entre humanos está na interação entre si, interação essa que pode assumir diversas maneiras e níveis. A comunicação é uma das maneiras às quais ocorre a troca de informações entre pessoas distintas de forma oral e jornais promovem o compartilhamento de informações visuais.

Sendo a troca de informação uma atividade básica entre os humanos, no ambiente do computador e das redes esse princípio não é diferente, mas o meio para sua realização diferentemente como acontece entre pessoas é realizada por meio de sinais digitais (FOROUZAN, 2009).

Em nível de computador, os dados são transitados de forma digital, ou seja, o estado do sinal pode assumir dois valores distintos, sendo 1 para alto e 0 para baixo, em outras palavras, um valor binário entre 0 e 1.

Na interligação entre redes prevista pela arquitetura TCP/IP, os dados digitais presentes no computador necessitam de serem transmitidos por um meio físico entre dois computadores distintos por meio de um sinal elétrico ou pelo próprio ar (FOROUZAN, 2009).

A camada física do modelo de referencia TCP/IP fica encarregada justamente de garantir a transmissão das informações entre redes e computadores por um meio cabeado como o cobre ou a fibra óptica ou sem fio como o *wireless* de forma a assegurar a imutabilidade na transmissão das informações (FOROUZAN, 2009).

2.1.2 Camada 2: Interface de rede

Como foi discutido no capitulo anterior, a camada física deve garantir que dois computadores possam se comunicar em um contexto de separação física e que deve ser garantida a entrega da informação da forma como foi enviada. Acontece que no mundo real, muitos são os fatores que podem acabar impedindo essa comunicação de acontecer, seja no rompimento de uma fibra ou no mal funcionamento de um equipamento *wireless*, é importante pensar que deva existir formas que detectem esses acontecimentos e que seja informado ao operador da rede a existência de dificuldades no caminho.

Na camada de rede, a existência de conexão física entre dois computadores garantida pela camada física é denominada como um enlace (KUROSE; ROSS, 2007). Dessa forma, em um ambiente como a Internet existem a presença de milhões de enlaces que conectam não somente os computadores entre si, mas também as redes em que esses computadores estão.

De forma a garantir a entrega da informação e de amparar a camada física, a camada de enlace possui mecanismos que fazem verificação dos quadros transitados pelos enlaces. Tanenbaum (2011) explica que um quadro ao percorrer um enlace e chegar ao destino é feito então um *feedback* a origem em relação ao sucesso do quadro, em caso de falha é feito uma solicitação a origem que encaminhe novamente o quadro para que seja completada a transmissão da informação (TANENBAUM, 2011). No contexto da camada de rede, Kurose e Ross (2007) explica que um dispositivo de rede é denominado como um nó.

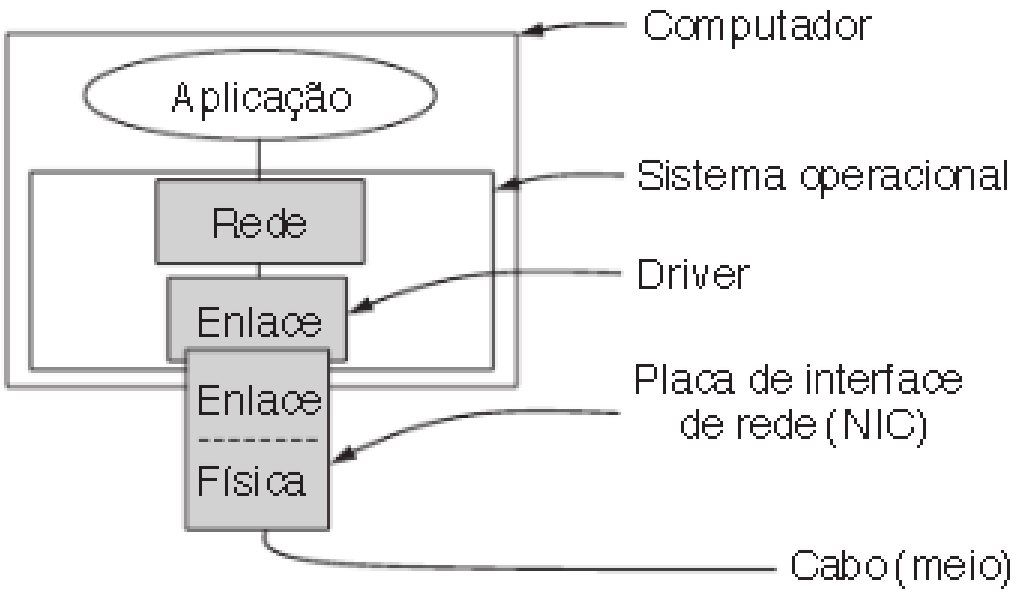
Não somente ajudando a camada física, a camada de rede tem a função de transmitir pelos enlaces os pacotes, denominados quadros quando inseridos no contexto da camada de rede, que são enviados da camada da Internet (KUROSE; ROSS, 2007). Para a ocorrência dessa transmissão, os quadros são transmitidos pelos enlaces, saltando por diversos nós e diferentes enlaces até seu destino que pode ser uma distancia física curta ou até mesmo do outro lado do mundo (KUROSE; ROSS, 2007).

Em um contexto de realidade, endereços são essenciais para que serviços como o correio ou entregas sejam realizados com sucesso, na internet esse conceito também é válido. Para que um *host* envie uma informação a outro *host*, desconsiderando a distancia que essa informação percorrerá pelos enlaces, é extremamente necessário o endereço do *host* de destino.

Os computadores possuem dispositivos de *hardwares* que possuem um endereço físico considerado único no ambiente de Internet. Este componente de hardware é denominado *Network Interface Card* (NIC) e é neste componente representado pela figura 2 que funcionam os serviços da camada física e de rede como destaca Tanenbaum (2011). É no NIC, comumente chamado de adaptador de rede, que está presente o endereço físico e tal endereço é

denominado como *Media Access Control* (MAC) (TANENBAUM, 2011).

Figura 2 – Representação da camada de enlace de dados e do adaptador de rede.



Fonte: (TANENBAUM, 2011)

Por fim, Kurose e Ross (2007) destacam que em paralelo ao endereço MAC existe o protocolo *Address Resolution Protocol* (ARP), sendo o objetivo desse protocolo a tradução do endereço IP em endereço MAC em decorrência da incompatibilidade entre eles. Tanenbaum (2011) explica que essa tradução é necessária pois o endereço MAC é disposto de forma hexadecimal e o endereço IP proveniente da camada de internet possui uma notação decimal.

2.1.3 Camada 3: Internet

Sendo a camada da Internet o próximo passo da pilha de protocolos TCP/IP, serão abordado nesta seção os serviços que fazem parte dessa camada.

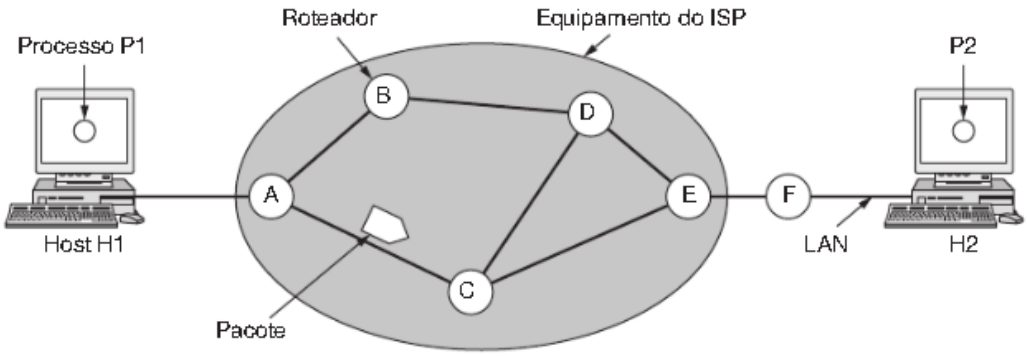
No serviço tradicional dos correios, uma encomenda é preparada e então enviada para o destinatário a partir de um caminho decidido nas filiais das transportadoras. A camada da Internet funciona de modo similar, sendo os roteadores desempenhando o papel de preparar e enviar um pacote de acordo com uma estrategia baseado no serviço de melhor esforço para o destino (KUROSE; ROSS, 2007).

O conhecimento acerca de todo o caminho do emissor ao destinatário, denominado por Comer (2016a) como um circuito virtual, é essencial para que a estratégia do menor caminho seja realizada, ou seja, a camada de Internet deve conhecer todo o conjunto de roteadores e enlaces (TANENBAUM, 2011).

As tabelas de roteamento presentes nos roteadores são os recursos repensáveis por determinar qual o caminho que o pacote percorrera e então realizar o repasse para a interface de saída escolhida (KUROSE; ROSS, 2007). Tanenbaum (2011) explica que uma interface de

saída é descrito como o próximo roteador para qual o pacote irar saltar, sendo esse comportamento descrito na figura 3 o contexto em que operam o princípio de roteamento na Internet.

Figura 3 – Representação do ambiente em que operam os protocolos da Internet.



Fonte: (TANENBAUM, 2011)

Assim como o objetivo dos serviços de correios é o transporte de encomendas ou cartas, no contexto da camada da Internet, o objetivo é o transporte de pacotes, denominados datagramas (KUROSE; ROSS, 2007). Tanenbaum (2011) descreve que para o funcionamento do roteamento na Internet, os datagramas movidos devem ter uma independência em relação aos roteadores percorridos pelo caminho, sendo esses datagramas conhecidos como *Internet protocol* (IP).

2.1.3.1 Endereço IP

O IP define um formato de datagrama que pode ser roteado pela Internet de forma que não sejam considerados os tipos de redes percorridas nem as diferenças entre os roteadores existentes (TANENBAUM, 2011).

Atualmente são previstos dois tipo de datagramas baseados no IP, o IPv4 considerado o mais comum e contendo um tamanho de cabeçalho de 32 bits e o IPv6 que foi criado como uma evolução devido ao esgotamento do IPv4 e contendo um tamanho de cabeçalho de 128 bits (COMER, 2016a).

Como foi descrito anteriormente nesta seção, no roteamento pela Internet, cada datagrama é direcionado a um roteador por uma interface de entrada e então repassado a uma interface de saída para ser roteado ao próximo roteador, Kurose e Ross (2007) defines essas interfaces como o endereço IP, podendo esse endereço ser o IPv4 ou o IPv6. Kurose e Ross (2007) terminam dizendo que para permitir uma conexão global através do endereçamento IP, o mesmo precisa ser único.

Os endereços IP são representados com uma notação decimal, dividido em duas partes, o prefixo para identificação de rede e o sufixo para identificar o *host* (COMER, 2016a).

Resumindo o IP, Comer (2016a) faz a seguinte definição:

"A cada *host* em uma internet IPv4, é atribuído um endereço exclusivo que é

> utilizado em toda a comunicação com o *host*. Para fazer o encaminhamento eficiente, um prefixo do endereço identifica uma rede e um sufixo identifica um *host* na rede."(Comer (2016a), p. 46).

Por fim, a camada de Internet deve lidar problemas de perda dos datagramas durante o roteamento. Quando foi dito que a camada de Internet oferece um serviço de melhor esforço, Kurose e Ross (2007) diziam que é oferecido um serviço onde não é feito nenhum tipo de confirmação de que os datagramas tenham chegado ao destino através do percurso entre os roteadores, sendo assim, é previsto um serviço com a finalidade de informar possíveis erros ocorridos durante o processo de roteamento, sendo esse serviço conhecido como *Internet Control Message Protocol* (ICMP) (COMER, 2016a).

2.1.4 Camada 4: Transporte

Kurose e Ross (2007) citam que por estar posicionada entre a camada da internet e a camada de aplicação, a camada de transporte exerce uma importância fundamental sobre a pilha de protocolos TCP/IP, desempenhando um papel importante de oferecer comunicação diretamente aos serviços de aplicação que estão sendo executados nos *hosts*.

Kurose e Ross (2007) destacam dois importantes serviços oferecidos pela camada de transporte:

1. Um serviço não orientado a conexão conhecido como *User Datagram Protocol* (UDP).
2. Um serviço orientado a conexão e que provê uma comunicação segura, sendo esse serviço conhecido como protocolo *Transmission Control Protocol* (TCP).

A camada de transporte segundo Tanenbaum (2011) é tida como núcleo do protocolo TCP/IP, oferecendo serviço de remessa de pacotes usando datagramas ou circuitos virtuais, se baseando na camada da Internet para oferecer o transporte de dados de um processo a outro em máquinas distintas oferecendo certo nível de confiabilidade.

A camada de transporte tem por objetivo fornecer uma comunicação lógica entre as aplicações de diferentes *hosts*, especificando melhor essa comunicação, deve ser levar em consideração que os *hosts* estejam diretamente conectados mas separados por uma distancia física (KUROSE; ROSS, 2007).

Tanenbaum (2011) explica que na camada de transporte, as conexões funcionam em três passos:

1. Estabelecer conexão
2. Transferência de dados.
3. Encerramento de conexão.

Tanenbaum (2011) diz que os serviços da camada de transporte e Internet são bastante similares mas apresentando uma diferença. Os protocolos da camada de rede funcionam

no ambiente de ISP, ou seja, funcionam nos roteadores e os protocolos da camada de transporte funcionam inteiramente nos *hosts*. Tanenbaum (2011) termina dizendo que embora as redes sejam suscetíveis a erros, a camada de transporte fornece um serviço confiável sobre a camada da Internet.

Uma outra diferença citada por (TANENBAUM, 2011) está na visibilidade das duas camadas pelos destinos, enquanto aplicações conseguem ter visibilidade sobre a camada de transporte, na camada de rede é somente visível para entidades de transporte.

Tanto na camada de Internet como na camada de transporte é utilizado o termo pacote no contexto de transporte de datagramas, Kurose e Ross (2007) explicam que na camada de transporte o termo pacote é denominado por segmento.

Kurose e Ross (2007) especificam que a camada de transporte funciona como uma extensão do serviço IP, ou seja, o serviço de entrega entre aplicações de dois *hosts*. Kurose e Ross (2007) seguem dizendo que o TCP e o UDP fornecem a verificação de integridade quando o serviço faz a integração de campos para detectar erros nos cabeçalhos dos segmentos. Kurose e Ross (2007) terminam dizendo que no protocolo UDP disponibilizado na camada de transporte apenas oferece serviços de entrega pelo fato do UDP ser um protocolo não orientado a conexões, ou em outras palavras, o UDP funciona semelhante ao IP onde os segmentos são propensos a se perderem pelo caminho.

Por outro lado, o TCP oferece uma entrega confiável de dados utilizando de controle de fluxo, temporizadores, controle de sequencia de segmentos e reconhecimento, fazendo com que o TCP ofereça um serviço que faz a entrega entre os *hosts* de forma confiável e em ordem além do TCP também garantir a verificação de integridade dos segmentos (KUROSE; ROSS, 2007). Kurose e Ross (2007) terminam dizendo que a partir dessas premissas o protocolo TCP age convertendo o serviço não confiável do IP entre *hosts* em um serviço confiável para aplicações entre *hosts*.

De acordo com Kurose e Ross (2007) o serviço de controle de congestionamento é presente não só no TCP mas como na Internet em geral. No TCP o controle de congestionamento permite que em enlaces congestionados, o TCP garanta o compartilhamento igualitário da largura de banda disponível ao fazer a regulagem da taxa com que o remetente envia o tráfego para o enlace. Em contramão, Kurose e Ross (2007) explicam que no UDP o tráfego de dados não é regulado de forma que uma aplicação usando o UDP é capaz de enviar dados sem limites de tempo ou velocidade.

2.1.5 Camada 5: Aplicação

Todas as camadas abaixo da camada de aplicação tem um objetivo, o transporte de dados entre os computadores, e todo esse trabalho prévio é direcionado para um simples objetivo, o objetivo de fornecer os dados para as aplicações presentes na quinta camada da arquitetura TCP/IP, a camada de aplicação (TANENBAUM, 2011).

São os serviços presentes na camada de aplicação que justificam todo o trabalho das camadas inferiores, desde serviços como o E-mail, o *streaming* de mídias, compartilhamentos

de arquivos P2P, o DNS e por último, as mensagens instantâneas que facilitam a vida de milhões de usuários, tudo isso implicando na camada de aplicação sendo indispensável ao mundo moderno e aos usuários (KUROSE; ROSS, 2007).

O princípio de funcionamento da camada de aplicação descrito por Kurose e Ross (2007) está no modelo cliente-servidor, onde um servidor tem por finalidade o fornecimento ininterrupto de serviços a Internet atendendo a requisições de usuários de todo o mundo. Kurose e Ross (2007) terminam conceituando a camada de Aplicação como protocolos que especificam detalhes acerca do formato e significado das mensagens que as aplicações podem compartilhar e também dos procedimentos que serão adotados durante a comunicação.

2.2 *Firewall* e Segurança

A capacidade de conexão de um computador em qualquer lugar, a qualquer outro computador pode ser uma prática perigosa, pois além do perigo de informações sigilosas vierem a público também tem o perigo do vazamento de informações dentro de instituições e empresas. Particularmente, a segurança de uma rede pode ser burlada por vírus e outras técnicas ilícitas para o roubo de dados digitais e fica a cargo do administrador da rede a tentativa de eliminar os prejuízos causados por um ataque a segurança da rede (TANENBAUM, 2011).

Segundo (NEUPANE; HADDAD; CHEN, 2018), os principais requisitos de segurança digital e seus objetivos resumidos estão listados abaixo:

1. Confidencialidade: Este item tem como objetivo a garantia de que informações confidenciais sejam limitadas apenas ao usuário autorizado.
2. Autenticidade: A garantia de que a identidade do sujeito ou recurso é a identidade reivindicada.
3. Integridade: Garantir ao menos que sejam feitas alterações autorizadas, que as informações sejam mantidas precisas e consistentes.

Atualmente, o número de usuários da Internet aumenta por minuto, segundo (STATS, 2016) o número de usuários ativos na internet no mundo no ano de 2000 era de 414.794.957 e em 2016 esse número passou para 3.424.971.237 o que representa um aumento considerável de usuários e atualmente estimam-se que cerca de 40% da população mundial tenha alguma conexão com a internet hoje.

Devido ao aumento expressivo no número de usuários conectados a Internet ao longo dos anos, aumentam-se também em 30% a probabilidade dos ataques causados pelo uso da Internet, para evitar estes tipos de ataques existem ferramentas IDS/IPS que tem como objetivo a detecção de anomalias na rede, dentre os quais podemos citar *snort*, *nmap*, *honeypot* e *firewalls*, sendo os *firewalls* a ferramenta mais usada para que organizações se protejam de ciberataques (SHARMA; PAREKH, 2017).

O *firewall* é uma coleção de componentes que são interpostos entre duas redes e que tem como objetivo filtrar o tráfego de dados entre as redes baseada em alguma regra de segurança, ou seja, o *firewall* serve como uma sentinela entre redes que são consideradas confiáveis de rede não confiáveis. Uma vez violada restrições de segurança, invasores podem se mover livremente dentro da rede (CHOWDHARY et al., 2018). Ainda segundo (BELLOVIN; CHESWICK, 1994), um *firewall* deve ter as seguintes propriedades:

- Todo tráfego deve passar pelo *firewall*, seja de dentro para fora ou vice-versa.
- Somente serão autorizados a passar pelo *firewall* o tráfego que for autorizado pela politica de segurança local.
- O *firewall* em si deve ser imune a falhas.

Kurose e Ross (2007) citam que os *firewalls* podem ser classificados em três categorias:

1. Filtro de pacotes
2. Filtros de estados
3. *Gateways* de aplicação.

2.2.1 Filtro de pacotes

O *firewall* por filtro de pacotes, funciona normalmente em um roteador que esteja na fronteira da rede interna com a ISP, de forma que todo o tráfego obrigatoriamente passa por esse roteador e esse roteador faz o filtro do que ele irá permitir e o que irá bloquear (KUROSE; ROSS, 2007).

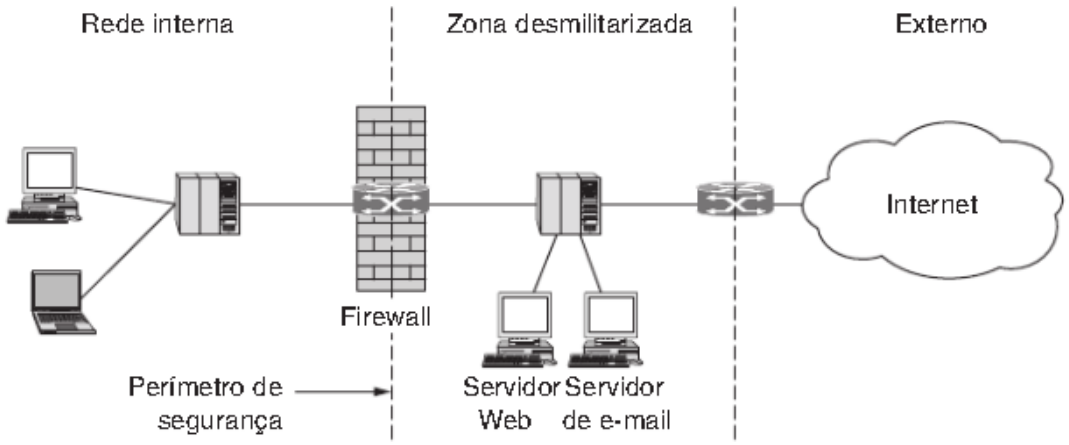
Tanenbaum (2011) explica que o *firewall* trabalha com regras que lista origens e destinos dos pacotes que entram na rede, podendo aceitar ou bloquear. Na pilha de protocolos TCP/IP, os pacotes são direcionados a uma porta e um endereço IP, sendo a porta o que identifica um serviço da camada de aplicação, como exemplo, a porta 80 é reservado ao HTTP e a porta 53 destinada ao DNS. A figura 4 exemplifica o processo do *firewall*.

Como não é interessante ao administrador bloquear toda a rede externa em relação a interna, então é previsto no esboço de *firewalls* as *Demilitarized Zone* (DMZ) como mostra a figura 4, que basicamente pode ser entendida como uma parte da rede interna onde todo o tráfego da internet é permitido consequentemente não sendo necessário o bloqueio total da intranet da internet.

Ainda em relação as estratégias de filtragens que o *firewall* permite, (KUROSE; ROSS, 2007) citam os seguintes:

- Endereço IP, podendo ser o IP do remetente ou IP destinatário.
- Tipo de protocolo designado no IP, podendo ser TCP, UDP etc.

Figura 4 – Ambiente de funcionamento do *firewall*



Fonte: (TANENBAUM, 2011)

- Porta TCP ou UDP.
- Protocolo ICMP.
- regras para pacotes transitando na rede.
- Regras para interfaces do roteador.

Comer (2016a) por fim diz que o filtro de pacotes deve trabalhar na ideia de que é mais fácil bloquear todos os pacotes e apenas permitir os pacotes destinados a redes específicas. Resumindo essa ideia, Comer (2016a) disse o seguinte:

> Para ser eficaz, um *firewall* que usa a filtragem de datagramas deve restringir o acesso a todas as origens IP, destinos IP, protocolos e portas de protocolo, exceto para aqueles computadores, redes e serviços que a organização decide tornar disponíveis externamente. Na verdade, um filtro de pacotes que permite a um administrador especificar quais datagramas serão admitidos, em vez de quais datagramas serão bloqueados, pode facilitar a especificação dessas restrições.(Comer (2016a), p. 421)

2.2.2 Filtro de estados

Comer (2016a) explica que a estratégia de bloquear todo o tráfego da intranet da internet cria um sério problema, isso impede que máquinas de dentro da intranet sejam impedidas de acessar serviços na internet. Sendo assim para que clientes de dentro da intranet sejam aptos a solicitar serviços da internet, a solução está no uso de *firewalls* com inspeção de estado ou *stateful*, que em sua essência realiza observações das conexões que são feitas de saída e então é feito uma adaptação para que sejam aceitos os pacotes de respostas.

Comer (2016a) ainda cita que esse tipo de *firewall* é útil para barrar a tentativa de iniciar conexões de dentro da intranet com qualquer destino fora dela, dessa forma o administrador exerce um controle maior sobre o fluxo do tráfego de pacotes.

O estado de uma conexão pode ser gerenciado de duas formas como explica Comer (2016a). Na primeira forma a conexão irá funcionar enquanto um tempo definido para *timeout* não tiver acabado, ou seja, as informações do estado serão removidas após a maquina detentora da conexão ficar inativa pelo tempo definido. A outra forma é através do monitoramento do fluxo da conexão até que ela esteja terminada.

Uma observação que Comer (2016a) faz, é que em conexões UDP, quando o fim da conexão não é previsto, a técnica por *timeout* no *firewall stateful* funciona como plano reserva para esses tipos de conexões.

2.2.3 *Gateways* de aplicação

No *firewall* de filtragem de pacotes, a principal técnica de inspecionamento como destacam Kurose e Ross (2007) é verificando de forma extensiva os cabeçalhos dos datagramas da camada de rede e transporte e que essa forma de filtragem apenas filtra o que entra na rede interna, de forma que clientes internos possam executar serviços da internet comprometendo a segurança da intranet. Acontece que em nível de aplicação, (KUROSE; ROSS, 2007) citam que as informações de identidade dos usuários são impossíveis de serem filtradas por esse tipo de *firewall*.

A junção das estratégias do *firewall* por filtro de pacote e *gateway* de aplicação permite a criação de uma forma mais refinada de *firewall*, onde não é feita apenas uma examinação dos cabeçalhos dos pacotes da camada de rede e transporte, mas sim uma observação a nível de aplicação onde é feita a verificação de todos os dados de uma aplicação bidirecional que passam pelo *firewall* (KUROSE; ROSS, 2007). Neste tipo de *firewall*, Kurose e Ross (2007) destacam que um mesmo *host* pode executar mais de um *gateway* de aplicação, e que cada um desses *gateways* funcionam de forma independente.

Infelizmente, os *gateways* de aplicação não são imunes a desvantagens, Kurose e Ross (2007) citam as principais desvantagens desse tipo de *firewall*:

1. A necessidade de um *gateway* de aplicação para cada tipo de serviço da camada de aplicação.
2. Comprometimento do desempenho, visto que pode acontecer de vários usuários terem de submeter seus dados ao mesmo *gateway* e gerando sobrecarga de trabalho.
3. Fica a cargo do *software* cliente especificar a conexão ao *gateway* e ainda dizer ao *gateway* a qual servidor se conectar.

Como exemplos de *gateways* de aplicação, Kurose e Ross (2007) citam *gateways* para os serviços de *Telnet*, HTTP, E-Mail e FTP.

2.3 Redes Definidas por Software

Tendo em vista o desenvolvimento da Internet, incluindo avanços em mobilidade, multimídia e a tendencia de uma sociedade virtual a gestão e configuração de redes tornaram-se

um processo altamente complexo, desafiador e que demanda muito tempo de um administrador de redes (MASOUDI; GHAFFARI, 2016).

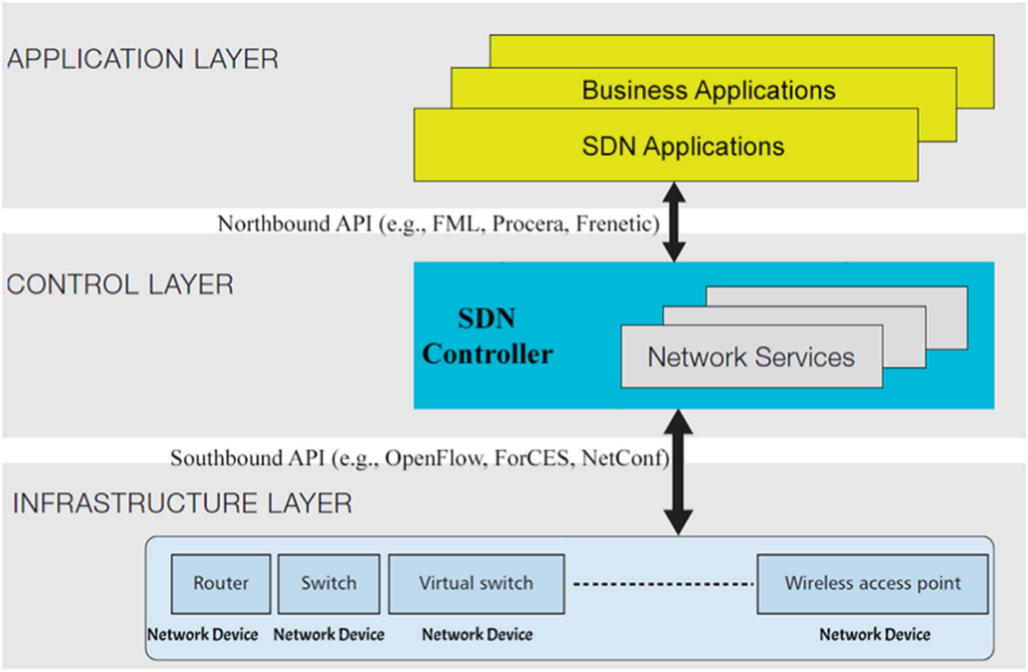
Redes de comunicação de dados geralmente consistem em dispositivos de usuário final ou *hosts* interconectados pela infraestrutura de rede e essa infraestrutura emprega elementos de comutação, como roteadores e *switches*. Tanto os roteadores como os *switches* possuem geralmente softwares "fechados"ou interfaces de controle que são específicas do fornecedor do dispositivo. Portanto devido a essa pluralidade em diferentes fabricantes e softwares, é bastante difícil para a infraestrutura de rede atual evoluir, ou resumidamente, a implantação de novos protocolos e serviços se torna uma tarefa árdua, um considerável obstáculo nas redes tradicionais (NUNES et al., 2014).

Deixando a situação ainda mais complicada, as redes atuais também são integradas verticalmente. O plano de controle (que controla o tráfego de rede) e o plano de dados (que encaminha o tráfego de acordo com as decisões tomadas pelo plano controle) são agrupados dentro dos dispositivos reduzindo a flexibilidade da rede (KREUTZ et al., 2015).

Com o aumento da demanda de aplicativos em tempo real, torna-se difícil dimensionar sem comprometer o desempenho das redes existentes em aspectos como confiabilidade e segurança. Os problemas de gerenciamento de rede são diversos e as alterações na infraestrutura são imprevisíveis, tornando um ambiente dinâmico bastante complicado para mudanças (SANDHYA; SINHA; HARIBABU, 2017).

A *Software Define Network* é um paradigma de rede emergente que dá a possibilidade para alterar as limitações das infraestruturas de redes tradicionais. Primeiro ele quebra a integração verticalizada dos dispositivos da rede, separando a lógica de controle da rede (plano de controle) do plano de dados. Segundo, com essa separação proposta, os *switches* de redes passam a ser simples dispositivos de encaminhamento e a lógica de controle é implementada em um controlador logicamente centralizado que implica em simplificação de aplicação de políticas e re-configuração, assim permitindo uma evolução da rede (KREUTZ et al., 2015). A Figura 5 apresenta a visão da arquitetura geral apresentada pelo paradigma SDN.

Figura 5 – Arquitetura SDN



Fonte: (ZHANG et al., 2018)

A SDN tem uma estrutura baseada em três partes principais como pode ser vista na figura 5. No nível mais baixo é composto pelo plano de dados onde é feito o encaminhamento do tráfego. No nível mais alto é composto pelo plano de aplicação onde é incluído os serviços de software, como ferramentas baseadas em protocolo simples de gerenciamento de rede (MASOUDI; GHAFFARI, 2016). Zhang et al. (2018) apresenta uma descrição mais detalhada sobre cada um dos componentes descritos na figura 5. Na camada mais inferior é onde são encontrados os dispositivos de encaminhamento, *switches*, roteadores e outros dispositivos similares. Esta camada lida prioritariamente com o encaminhamento dos fluxos de dados, que são gerenciados pela camada superior, ou seja, as decisões são tomadas nos controladores de rede e as regas dos controladores são carregadas nestes dispositivos (ZHANG et al., 2018).

Uma observação a ser feita é que na infraestrutura, todos os dispositivos se comportam igualmente e são referenciados como *switch* OpenFlow, não tendo distinções entre *switches* e roteadores (MASOUDI; GHAFFARI, 2016). Na seção 2.3.1 é apresentada a arquitetura do *switch* OpenFlow e a explicação de seus componentes.

Intermediando a camada de infraestrutura e controle é encontrado o *Southbound interface* (SBI). O SBI é descrito por Zhang et al. (2018) como interfaces de comunicação que tem como objetivo a facilitação da comunicação entre os dispositivos da camada de infraestrutura com a camada de controle. Comunicação com o controlador SDN, implementação de políticas sobre o gerenciamento da rede, controle sobre todos os dispositivos da camada inferior e descoberta da topologia estão entre os principais objetivos que o SBI tem em mãos. O OpenFlow, descrito na seção 2.3.1, é classificado como uma interface SBI, sendo ele o mais popular em uso atualmente.

O controlador da rede SDN, onde todas as decisões pertinentes ao encaminhamento e status da rede, se encontra na camada de controle, podendo ser composta por um ou vários controladores. As atividades dos controladores são resumidas em coletar informações e prover de todo o conhecimento da rede para ser capaz de providenciar as regras para os dispositivos da camada inferior (ZHANG et al., 2018). Na seção 2.3.2 se encontra uma descrição da arquitetura de um controlador.

A comunicação entre a camada de aplicação e a camada de controle é realizada pelo *Northbound interface* (NBI). A NBI é descrito por Zhang et al. (2018) como um conjunto de API*s* que provê a comunicação com o controlador e consequentemente permitir aos aplicativos da camada de aplicação meios de controlar o gerenciamento dos recursos da rede.

Na camada superior da arquitetura SDN, se encontram as aplicações e serviços SDN para a manipulação da rede sob poder do usuário. Zhang et al. (2018) explica que é nesta camada que são feitas as comunicações com as demais, graças a interfaces que facilitam a movimentação de dados entre as camadas.

Comer (2016a) explica que o paradigma de SDN surgiu como uma solução híbrida ao combinar a tecnologia de redes de conexão orientada e da sobreposição de roteamento. As redes orientada a conexão são tidas como redes que têm a capacidade de definir rotas de encaminhamentos diferentes para cada fluxo de pacotes que trafegam por elas.

A sobreposição de roteamento é tido como uma técnica onde é feito a criação de uma topologia de rede virtual, onde a partir do encaminhamento existente. é realizado a entrega dos pacotes na rede virtual. Essa definição como destaca Comer (2016a) se assemelha a técnica de tunelamento de forma que os roteadores enxerguem os túneis como conexões ponta-a-ponta dos protocolos de roteamento.

Conforme Comer (2016a) destaca que cada um desses métodos tem seus pontos fonte e fracos. As redes orientadas a conexão podem ser implementadas por *hardwares* oferecendo assim o uso de uma alta velocidade de classificação baseado no hardware implementado e nas técnicas de comutação de rotulo, de forma que essas redes podem ser facilmente escaladas para redes mais velozes. Por fim Comer (2016a) termina dizendo que sobreposição de roteamento tem a vantagem de serem flexíveis e apresentarem pouca dificuldade na mudança por se basear em software.

Pensando nas características apresentadas pelas redes orientada a conexão e na técnica de sobreposição de roteamento, Comer (2016a) explica que o paradigma de SDN surgiu da combinação dos pontos fortes dessa técnica tendo em vista os seguintes objetivos.

- A partir da classificação de alta velocidade do hardware evitar a sobrecarga produzida pelo uso do software.
- Usar o hardware para o encaminhamento de alta velocidade evitando o uso do software para tal ação devido a menor velocidade.
- Dar poder aos gerentes de rede para escolherem como será feito o roteamento dos pacotes gerando uma maior confiabilidade ao evitar os protocolos de roteamento padrão.

- Permitir que a internet seja dimensionada por aplicativos de gestão como configuração dos dispositivos envolvidos, tirando esse poder das mãos humanas.

2.3.1 OpenFlow

Como foi discutido na seção anterior, o paradigma de SDN se prova bastante útil ao quebrar o forte acoplamento dos *switches*, então tornando o plano de controle programável. Acontece que a realização de todas essas vantagens partiram de uma ideia que nascia no ano de 2008 sendo chamado de OpenFlow

O protocolo OpenFlow primeiramente foi proposto com o objetivo de possibilitar experimentos para o controle de uma rede local. Um dos experimentos realizados possibilitava o controle de encaminhamento de pacotes através de regras definidas por *software* (MCKEOWN et al., 2008). Comer (2016a) explica que o protocolo OpenFlow passou a ganhar mais popularidade quando houve uma aceitação de fabricantes de dispositivos de rede para o seu uso

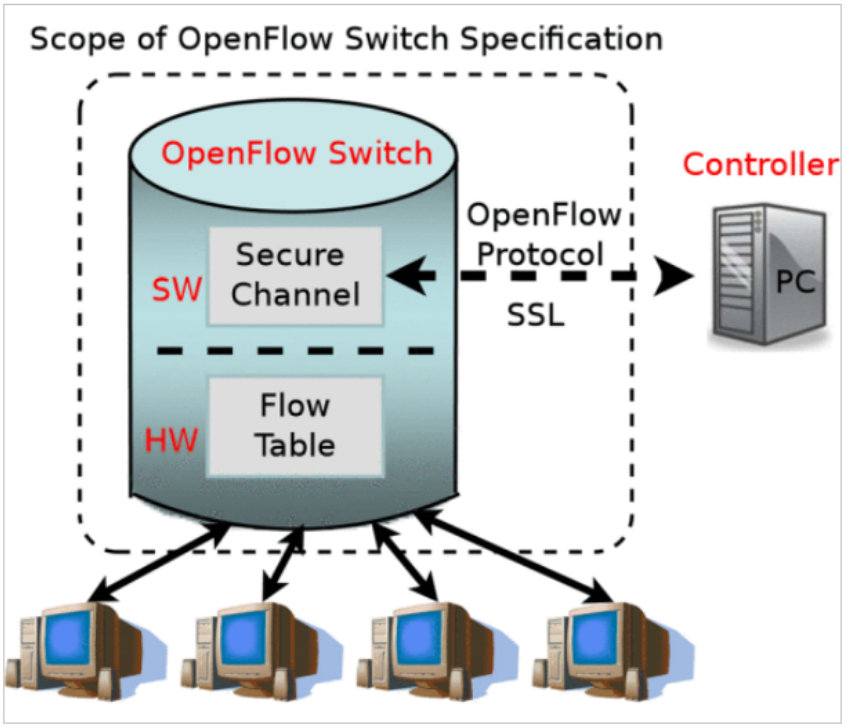
A importância do OpenFlow para o paradigma de SDN é justamente por ser o facilitador da funcionalidade da SBI, que é basicamente o módulo que controla o plano de controle desvinculado do plano de dados presente nos *switches* e permitindo o seu controle pelo gerenciador de redes (COMER, 2016a). A figura 6 descreve a arquitetura do OpenFlow.

Como ilustra a figura 6, o OpenFlow oferece serviços de manipulação da tabela de fluxo dos *switches* como inserir, excluir, modificar as regras de acordo com a demanda do ambiente de redes, tudo programável em um controlador por meio de um canal TCP seguro e remotamente (na figura 6 é usado o protocolo SSL para a realização dessa conexão segura) (XIA et al., 2015).

Bholebawa, Jha e Dalal (2016) complementa que nas redes tradicionais, todos os comutadores envolvidos em uma ISP se limitam basicamente na recepção dos pacotes e na decisão de encaminhamento se baseando na regra de roteamento disponíveis nos cabeçalhos dos pacotes, diferentemente de uma rede OpenFlow onde os mesmos comutadores se concentram apenas no encaminhamento dos pacotes com as regras que são adicionadas em sua tabela de fluxo por um controlador centralizado que faz o gerenciamento de todos os comutadores disponíveis na rede. Yazdinejad, Bohlooli e Jamshidi (2018) terminam explicando que uma tabela de fluxo em um *switch* OpenFlow na versão 1.3.0 possui cinco campos, sendo eles:

- "Cabeçalho": O cabeçalho tem como objetivo a correspondência de pacotes recebidos com informações da tabela de fluxo com base em informações de endereço IP, MAC e ainda portas TCP ou UDP.
- "Ação": Com o objetivo de definir ações de processamento sobre o pacote com base em políticas definidas no controlador.
- "Estatística": Juntar estatísticas do fluxo de tráfego ocorrendo no *switch*.
- "Prioridade": Diz respeito a prioridade de correspondência de fluxos.

Figura 6 – *Switch* OpenFlow. Arquitetura básica do OpenFlow.



Fonte: (MCKEOWN et al., 2008)

- "Tempo limite": Especifica o tempo limite que um fluxo pode ficar desatualizado
- "*Cookie*": Uma chave única que os controladores usam para manipulação de fluxos.

Ainda em respeito ao cabeçalho de um fluxo OpenFlow, A tabela 1 sintetiza os campos presentes de acordo com as especificações de um *switch* OpenFlow 1.3.0.

Tabela 1 – Campos presentes em um cabeçalho OpenFlow

Campo	Especificação
IN_PORT	Diz respeito a uma porta física ou lógica
ETH_DST	Endereço de destino Ethernet
ETH_SRC	Endereço de origem Ethernet
ETH_TYPE	Tipo de carga útil do pacote OpenFlow para Ethernet
IP_PROTO	Especifica o tipo de protocolo IP, podendo ser IPv4 ou IPv6
IPV4_SRC	IPv4 de origem
IPV4_DST	IPv4 de destino
IPV6_SRC	IPv6 de origem
IPV6_DST	IPv6 de destino
TCP_SRC	Porta TCP de origem
TCP_DST	Porta TCP para destino
UDP_SRC	Porta UDP de origem
UDP_DST	Porta UDP para origem

Fonte: (YAZDINEJAD; BOHLOOLI; JAMSHIDI, 2018)

Comer (2016b) explica que OpenFlow não se categoriza como um padrão regularizado pelo IETF, mas sim uma especificação que pode ser encontrado *online*[1] e que é mantido pelo Consórcio OpenFlow[2].

Como o OpenFlow é um projeto *OpenSource*, sua disponibilidade é por versões. A primeira versão documentada no Consorcio OpenFlow é a 0.2.0 datado de 2008 onde foi especificado o *switch OverFlow* e atualmente se encontra na versão 1.5.1. Na tabela 2 são referenciadas por Zhang et al. (2018) as principais versões do OpenFlow.

Tabela 2 – Versões do OpenFlow

Versão OpenFlow	Data de liberação
1.0.0	12/2009
1.1.0	02/2011
1.2.0	12/2011
1.3.0	06/2012
1.4.0	10/2013
1.5.0	12/2014

Fonte: (ZHANG et al., 2018)

Ainda sobre as principais versões do OpenFlow, Zhang et al. (2018) discute as principais mudanças e evoluções entre elas. Na versão 1.0.0 foram adicionadas além da especificação do *switch OpenFlow* o suporte para definição de tabela de fluxo e manipulação de endereços IP e MAC. Continuando o processo de desenvolvimento na versão 1.1.0 foram adicionados a capacidade para múltiplas tabelas de fluxo, conjunto de ações como controle de perda de conectividade dentre outras melhorias, como portas virtuais. Em sua versão 1.2.0 foi adicionado suporte ao padrão de IPv6 e o suporte a múltiplos controladores. Na versão 1.3.0 teve a adição de manipulação da extensão do *header* IPv6, suporte mais flexível a tabelas e por fim uma considerável adição ao permitir o controle de congestionamento de fluxo. A versão 1.4.0 trouxe suporte a portas ópticas, monitoramento de fluxo, tabelas sincronizadas e um mecanismo de gerenciamento de lotação de tabelas de fluxo. Por fim, a versão 1.5.0 incluiu melhorias como um *pipeline* que reconhece diferentes tipos de pacotes, não somente o *Ethernet*, além de uma técnica de tabela de saída que permite com que o pacote transitado seja baseado no contexto de porta de saída.

Discutidos as versões do OpenFlow, são apresentadas na tabela 3 um resumo das versões e suas melhorias.

2.3.2 Controlador

Como foi discutido nas seções anteriores, o OpenFlow permite que os diversos *switches* existentes em uma rede possam ser controlados por um controlador centralizado, os autores Masoudi e Ghaffari (2016) destacam que por esse papel de muita importância, o contro-

[1] https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf
[2] https://www.opennetworking.org

Tabela 3 – Resumo das versões do OpenFlow

Versão OpenFlow	Características
1.0.0	Especificação do *switch* OpenFlow, tabela de fluxo, suporte a IP e MAC
1.1.0	Múltiplas tabelas de fluxo, portas virtuais e Conjunto de ações
1.2.0	Suporte ao IPv6 e suporte a múltiplos controladores
1.3.0	Extensão do *header* IPv6, controle de congestionamento e flexibilidade para tabelas de fluxo
1.4.0	Portas ópticas, monitoramento de fluxo, sincronização de tabelas e gerenciamento de lotação das tabelas
1.5.0	Tabela de saída e pipeline de pacotes

Fonte: (ZHANG et al., 2018)

lador em si é tido como o componente mais valioso no paradigma SDN, afetando diretamente o desempenho da rede.

Tendo todo um papel de destaque, Bholebawa e Dalal (2018) destacam que o controlador SDN é similar a um sistema gerenciável, sendo responsável por gerenciar toda a lógica de funcionamento da rede SDN. Masoudi e Ghaffari (2016) citam em seu estudo que a arquitetura do controlador é baseado em duas partes. Na primeira parte, o controlador dispõe das aplicações que são usadas em uma rede SDN, como um *firewall*, e a segunda parte é tida como o sistema gerenciador da rede, ou *Network Operating System* (NOS), onde esse sistema em si será responsável por controlar a rede. A figura 7 apresenta a arquitetura discutida por Masoudi e Ghaffari (2016).

A tabela 4 apresenta os principais controladores SDN disponíveis online e logo em seguida, são apresentados algumas observações a cerca desses controladores.

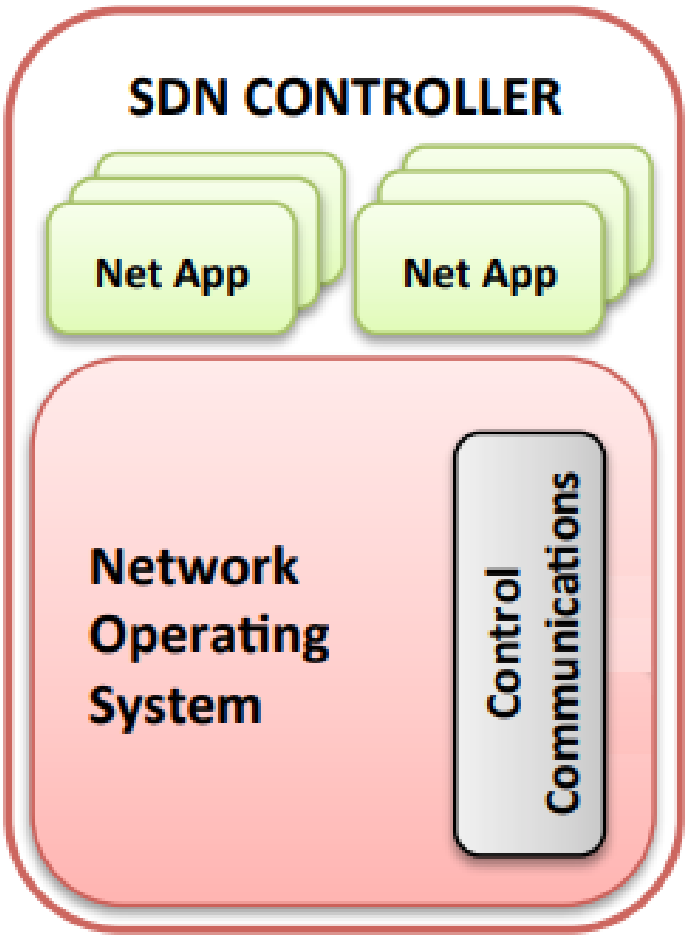
Tabela 4 – Apresentação dos principais controladores SDN

Controlador	Implementação	Código abeto	Desenvolvedor
POX	Python	Sim	Nicira.
NOX	Python/C++	Sim	Nicira
MUL	C	Sim	Kucloud
Beacon	Java	Sim	*Stanford University*
Helios	C	Não	NEC.
Floodlight	Java	Sim	BigSwitch.

Fonte: Adaptado de (MASOUDI; GHAFFARI, 2016)

A respeito dos controladores listados na tabela 4, Masoudi e Ghaffari (2016) ainda adicionam as seguintes informações. O NOX é tido como o primeiro controlador SDN baseado no protocolo OpenFlow, sendo atualmente escrito em C++. POX é um controlador baseado no OpenFlow e conhecido por oferecer rápido desenvolvimento e prototipagem, Bholebawa e Dalal (2018) complementa dizendo que o POX tem uma vantagem sobre o NOX em velocidade. MUL é um controlador com suporte ao OpenFlow 1.3, além de oferecer uma infraestrutura

Figura 7 – Arquitetura do controlador OpenFlow.



Fonte: Adaptado de (KREUTZ et al., 2015)

multi-threading. Beacon é um controlador multi-plataforma e modular, também oferece operação por *threads* e baseada em eventos. O controlador Helios é um controlador que oferece uma interface Shell programável e suporte ao OpenFlow. Por fim, o Floodlight é um controlador baseado no Beacon que oferece um suporte a *switches* físicos e lógicos.

3 Trabalhos relacionados

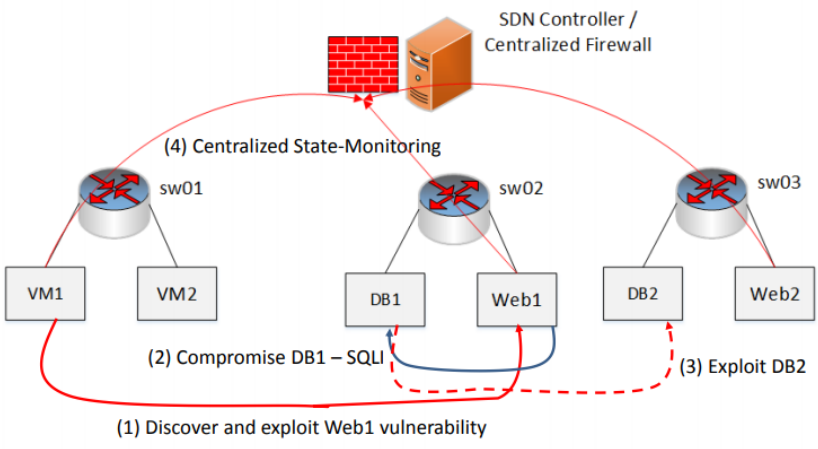
Nesta seção serão discutidos os principais trabalhos científicos relacionados a natureza do problema de pesquisa deste artigo.

Para o primeiro trabalho relacionado a este estudo, Chowdhary et al. (2018) mostra que em um cenário usando uma arquitetura de *firewall* centralizada aplicado em um ambiente de data centers as regras de fluxo podem ser usadas para bloquear o tráfego de um segmento, acontece que em ataques originados no plano de dados que dependem de informações de conexão o *firewall* centralizado não tem a capacidade de detectar tal ação.

Chowdhary et al. (2018) levanta em seu trabalho dois casos de exemplo, a figura 8 ilustra esses dois casos, no primeiro caso é considerado um cenário onde é feito o uso de uma arquitetura *firewall stateless* (sem controle de conexão) centralizado. Neste cenário é proposto o uso de 3 *switches* OpenFlow conectados em um controlador SDN centralizado com a funcionalidade *firewall*. Supondo que o tráfego entre determinadas VM's através da rede foi permitido, se houver uma vulnerabilidade de segurança na Web1, DB1 e DB2, e ainda o invasor estiver localizado na VM1, mesmo que o tráfego entre VM1 e Web1 esteja permitido o *firewall stateless* não pode inspecionar o estado de conexão da rede.

No segundo caso de exemplo Chowdhary et al. (2018) propõe o uso de um *firewall stateful* (com controle de conexão) centralizado. Neste cenário, se o controlador SDN for inspecionar cada pacote trafegado pela rede, consequentemente, todo o tráfego da rede terá de passar por esse controlador, como mostra na figura 8. A decisão dessa ação pode acarretar uma rápida sobrecarga sobre o controlador SDN.

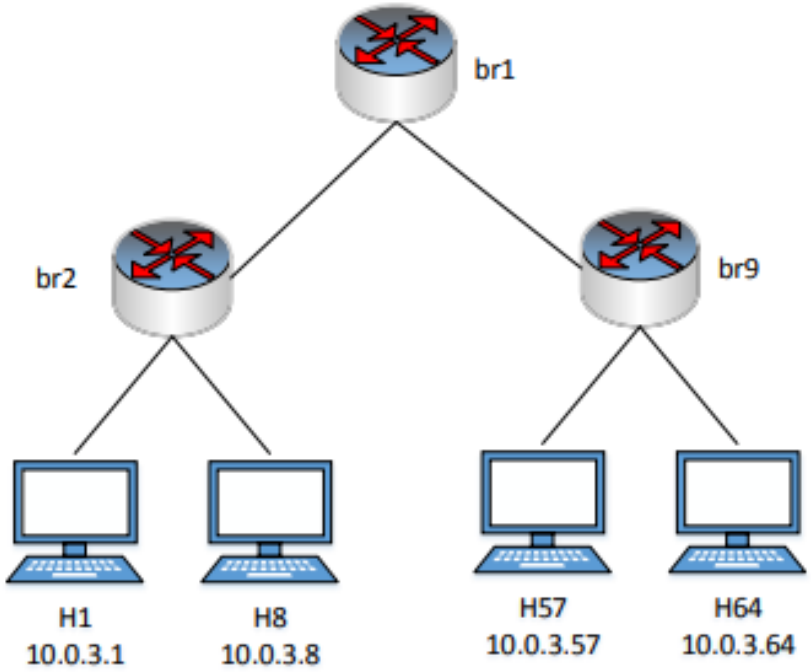
Figura 8 – Problemas de segurança em um *firewall* centralizado



Fonte:Adaptado de (CHOWDHARY et al., 2018)

Assim o autor com base nesses dois casos de teste propõe o uso de uma arquitetura de *firewall stateful* distribuída com uma topologia construída em forma de arvore para que cada *switch* possa rastrear localmente os eventos dos *hosts* a ele destinado conforme a figura 9 mostra.

Figura 9 – Topologia proposta por Chowdhary et al. (2018)



Fonte:Adaptado de (CHOWDHARY et al., 2018)

O trabalho de Suh et al. (2014) se concentrou no desenvolvimento de uma aplicação de *firewall* utilizando um controlador SDN juntamente com o protocolo OpenFlow com o principal objetivo de investigar o uso de módulos de *firewall* baseados em software sem a necessidade do uso de um hardware dedicado. O estudo de Suh et al. (2014) foi conduzido sobre o controlador POX utilizando o Mininet para a virtualização da topologia obtendo os resultados esperados para as regras implementadas no *firewall*.

No trabalho realizado por Pena e Yu (2014), são explorados as possibilidades de segurança no desenvolvimento de um *firewall* de filtro de pacotes distribuído que utilize das vantagens disponibilizadas pelo paradigma SDN e uma relação entre a latência do *firewall* desenvolvido em relação aos *firewalls* tradicionais. O desenvolvimento do trabalho foi feito sobre o controlador POX e utilizando do simulador Mininet, como testes para validar as funcionalidades do *firewall*, foi utilizado o comando ping para verificar a precisão das regras de filtro cadastradas. Os resultado obtidos por Pena e Yu (2014) foi de alta precisão na filtragem dos pacotes baseados nas regras carregadas no controlador e no teste de latência Pena e Yu (2014) chegou a conclusão que o uso do *firewall* distribuído não apresentou nenhum resultado negativo.

A *Internet of Things* (IoT) tem representado atualmente um grande desafio devido ao grande aumento de dispositivos com capacidade de conexão a internet e a geração excessiva de dados e uma das possibilidades geradas pela IoT é a das casas automatizadas. Seguindo essa direção, é proposto e simulado por Pessoa e Duarte-Figueiredo (2017) uma plataforma para o gerenciamento de casas automatizadas utilizando NodePI juntamente com o paradigma de SDN. Juntamente ao objetivo do trabalho e pensando na segurança da rede envolvida foi utilizado um *firewall* de aplicação baseado no controlador POX para investigar a segurança dos dispositivos usados na plataforma, chegando em um resultado onde dispositivos não autorizados estavam impedidos de acessar a rede.

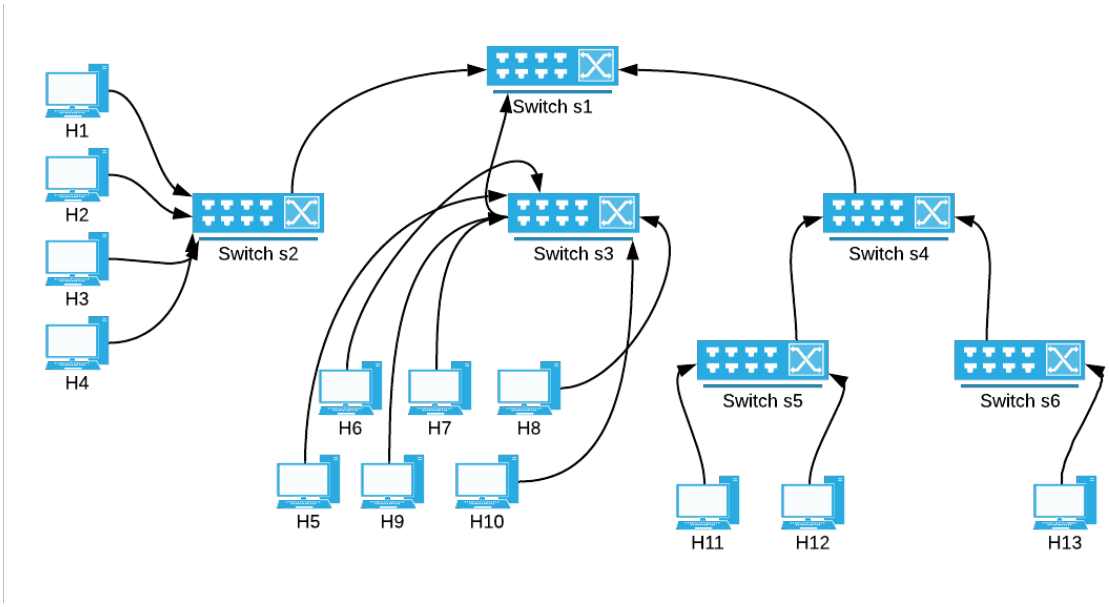
4 Materiais e Métodos

Neste capítulo foram discutidas a escolha das principais ferramentas e itens para a realização dos objetivos do trabalho. Também foram apresentados nesta seção o cenário para a simulação do trabalho e apresentando as regras ao qual este trabalho se baseou. Na seção

4.1 Definição do cenário, Simulador e o Controlador SDN

Para atender os requisitos do trabalho, foi necessário a elaboração de um cenário de rede para as simulações que apresentasse uma topologia semelhante a topologia utilizada na instituição de ensino. A figura 10 ilustra a topologia utilizada nas simulações, baseada em uma estrutura hierárquica de várias redes e sub-redes.

Figura 10 – Cenário proposto para realização do trabalho



Fonte: Elaborado pelo Autor

O simulador utilizado nos experimentos da proposta foi o Mininet. Ele é uma ferramenta *opensource*, desenvolvida com intuito de simular redes SDN e é encontrada facilmente na web. A escolha desse simulador foi motivada pelo fato ser uma ferramenta com uma vasta documentação para consulta e uma comunidade ativa de pessoas que conhecem apoiam o projeto. O Mininet é baseado na linguagem *Python* e é disponibilizado em uma VM na web [1]. Existem ainda disponível na web outras opções de simuladores de redes SDN além do Mininet, como:

- EstiNet

[1] URL:http://mininet.org/download/

- NS-3

O Estinet é um simulador que apresenta uma abordagem especifica para testar funções e desempenhos de controladores OpenFlow além de combinar as vantagens das abordagens de simulação e emulação e prover a cada *host* simulado o poder de executar o sistema Linux e qualquer programa de aplicação real baseado em Unix. Porém, esse simulador possuiu uma quantidade de material de consulta limitada, além de necessitar adquirir uma licença para o seu uso.

O simulador NS (*Network Simulator*), permite a avaliação de desempenho de uma rede em termos de tráfego e desempenho. A terceira versão desse simulador, a NS-3, permite a simulação de redes OpenFlow. Contudo, o NS-3 apresenta algumas limitações, o módulo OpenFlow não simula com precisão o funcionamento real de uma rede OpenFlow,pois não possui suporte a tráfego TCP entre *switch* e controlador e não possui suporte ao STP. O NS-3 possui pouca documentação disponível para as simulações com os módulos OpenFlow, dificultando a aprendizagem e implementação.

Junto com a ferramenta de simulação foi escolhido o controlador de rede SDN. A escolha do controlador foi feita seguindo as mesmas escolhas do Mininet, foi levado em consideração a disponibilidade do código fonte e o suporte disponível na web. A tabela 5 apresenta os principais controladores disponíveis, sendo eles especificados pela implementação, disponibilidade de código fonte e uma apresentação de sua visão global.

Tabela 5 – Apresentação dos principais controladores SDN

Controlador	Implementação	Código abeto	Visão global
POX	Python	Sim	Controlador geral de SDN em código aberto escrito em Python.
NOX	Python/C++	Sim	O primeiro controlador OpenFlow escrito em Python e C++
MUL	C	Sim	Controlador OpenFlow que possui uma infraestrutura multissegmentado baseada em C.
Beacon	Java	Sim	Controlador OpenFlow modular, baseado em Java e que suporta operações baseadas em eventos e encadeadas
Helios	C	Não	Um extensível controlador OpenFlow baseado em C que fornece uma classe programável para executar experiências integradas.
Floodlight	Java	Sim	Controlador OpenFlow baseado em Java, baseado na implementação do Beacon, que fornece um funcionamento com *switches* físicos e virtuais do OpenFlow.

Fonte: Adaptado de (MASOUDI; GHAFFARI, 2016)

Diante desta tabela e das especificações de cada controlador e de características como disponibilidade do código fonte, suporte e linguagem de escrita, o Floodlight foi a escolha mais condizente para a realização deste trabalho. O Floodlight é um controlador SDN, baseado na

linguagem Java tendo como requisitos:

1. Eclipse IDE
2. Linguagem de programação Java

Para utilização do simulador Mininet foi utilizado VM, sendo emuladas pelo programa VirtualBox e o computador hospedeiro possui as seguintes configurações de *hardware*:

- Notebook contendo as especificações:
  - Processador: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
  - Memória RAM: 8GB;
  - SO: Windows 10 Pro (SO de 64 bits)
  - Armazenamento: Hard Drive 1 TB, SATA (5400 RPM)

A máquina virtualizada possui sistema operacional Ubuntu e as seguintes configurações:

- Processador: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz (Compartilhando da maquina hospedada).
- Memória RAM: 4GB (Compartilhando da maquina hospedada);
- SO: Ubuntu 14.04 (SO de 64 bits);
- Armazenamento: Hard Drive 8 GB, (Compartilhado da maquina hospedada).

Em posse desses materiais, foi possível a realização de todo o desenvolvimento como pode ser visualizado na seção 5.

4.2 Definições de regras de *firewall*

Dentre as opções de bloqueio disponível no módulo do *firewall* de aplicação presente no controlador Floodlight, foram escolhidas as seguintes regras para aplicação neste trabalho.

- Bloqueio de endereço MAC
- Bloqueio de endereço IP
- Bloqueio de porta TCP

Na documentação presente na página do controlador Floodlight foi destacado que funciona ocorre através de requisições para uma *Rest API*. Logo para o funcionamento adequado foi necessária o uso da ferramenta Curl, sendo essa ferramenta que viabiliza a realização das requisições para o *firewall* do controlador.

5 Desenvolvimento

Este capítulo foi dividido em 4 partes: a seção 5.1 apresenta o processo de inicializar o Floodlight. A seção 5.2 caracteriza por apresentar o Mininet e o processo de criação de topologias. Na Seção 5.3 é criado a topologia onde será simulado o trabalho. E finalmente na seção 5.4 é mostrado os passos para carregar o módulo do *firewall* juntamente com as regras padrão na topologia em simulação.

5.1 Instalação do controlador

Conforme o estudo realizado por Jesus et al. (2016), o controlador Floodlight é recomendado para redes que empregue a topologia *tree*, por consumir menos memória e menos tempo para criar e inicializar a rede SDN. Essas características viabilizaram a utilização e escolha do Floodligth como o controlador para realizar as simulações e a elaboração da proposta do *firewall*.

O Código 1 detalha os comandos utilizados na instalação e execução do controlador Floodlight. Ressaltando que para execução dos comandos é necessário o privilégio de *root* no sistema operacional.

```
1 root@Floodlight:/home/Floodlight# cd Floodlight
2 root@Floodlight:/home/Floodlight# ant
3 root@Floodlight:/home/Floodlight# java -jar target/Floodlight.jar
```

Código 1 – Construindo e compilando o controlador Floodlight

Após a compilação do código fonte para instalação do controlador, são realizadas as configurações de endereço IP e porta do controlador, editando o arquivo floodlightdefault.properties que se encontra no diretório /home/Floodlight/src/main/resources, modificando as linhas apresentadas no código 2.

```
1 net.floodlightcontroller.core.internal.FloodlightProvider.openFlowPort
    =6653
2 net.floodlightcontroller.core.internal.OFSwitchManager.openFlowPort=6653
3 net.floodlightcontroller.core.internal.OFSwitchManager.openFlowAddresses
    =0.0.0.0
```

Código 2 – Configurações padrões para ip e porta para inicialização do controlador

5.2 Criação de topologias no Mininet

O Mininet é bastante útil para a simulação de topologias de redes, permitindo criar diversos modelos e possibilitando ao desenvolvedor simular diferentes cenários. O comando 3 iniciou a topologia miníma, sendo ela constituída de um *switch* OpenFlow conectada a dois *hosts* e um controlador.

```
1 sudo mn
```

Código 3 – Comando para inicialização de topologia no Mininet

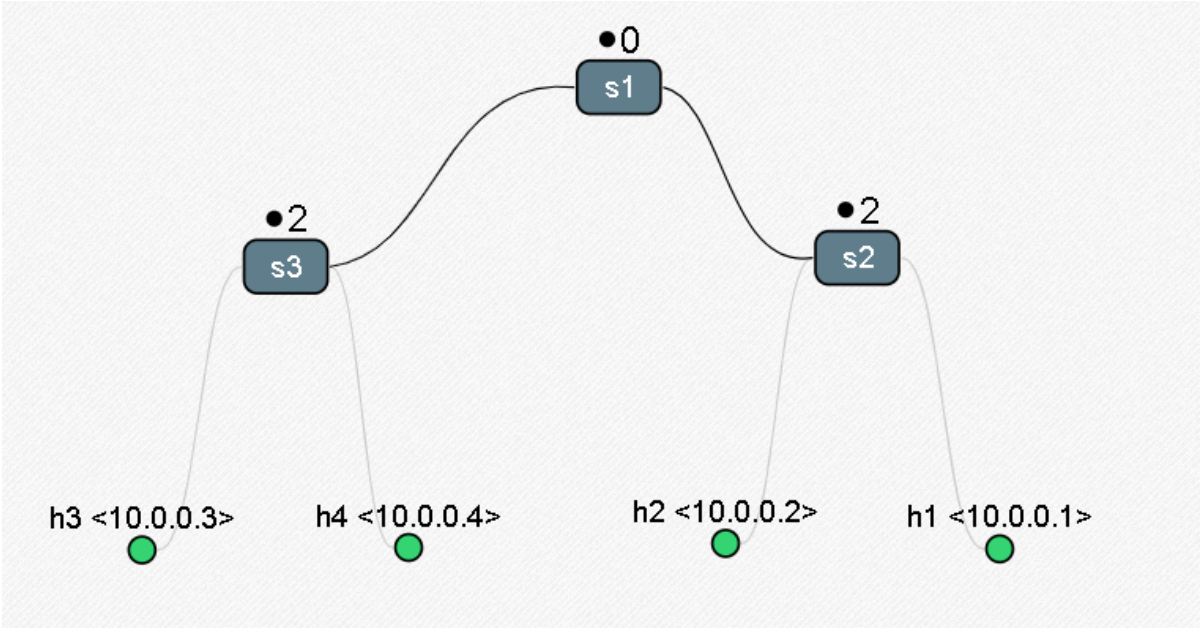
O parâmetro --topo"é utilizado para a definir a topologia da rede, como exemplificado no comando 4.

```
1 sudo mn --topo=tree,2
```

Código 4 – Inicialização de uma topologia com estilo árvore no Mininet

O código 4 inicializou uma topologia de rede em forma de árvore, a figura 11 ilustra essa topologia criada. Pela figura 11 foi constatada a criação de um ambiente de rede composto por 3 *switches*, 4 *hosts* e 1 controlador. Vale lembrar que o Mininet não oferece recursos gráficos para a visualização da topologia criada.

Figura 11 – Representação gráfica da topologia criada.



Fonte: Elaborado pelo Autor

Para a visualização da figura 11 foi necessária o uso de uma ferramente externa ao Mininet. Para isto foi utilizado a ferramenta *Spear*[1]. Tal ferramenta que pode ser encontrada online necessita apenas das informações de nódulos que pode ser obtida através do comando "*dump*"e das listas de *links* através do comando "*links*", ambos os comandos digitados no Mininet.

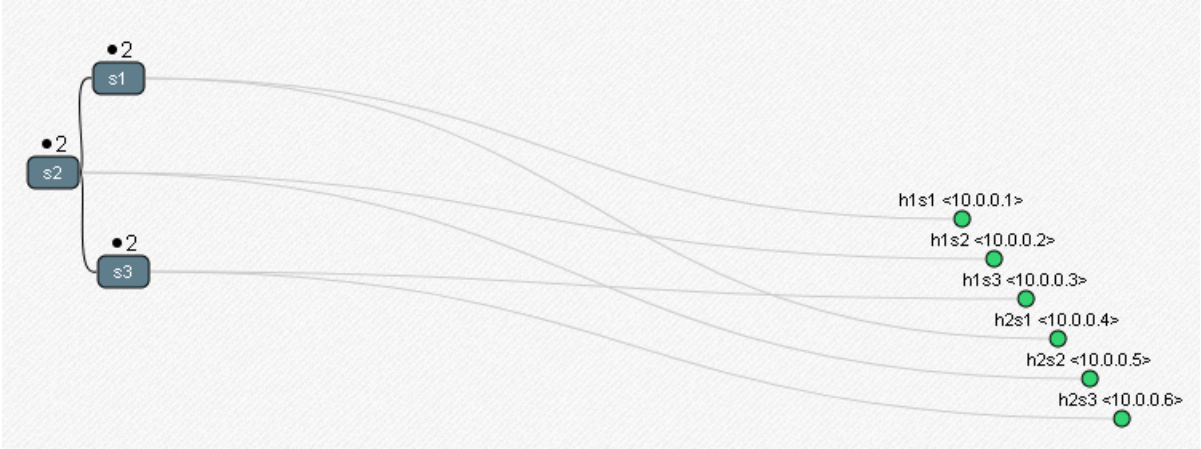
Ainda sobre criação de topologias padrões com o comando 4, o Mininet permite também ao usuário a mesma construção utilizando os parâmetros "linear"e "single". Segue uma explicação dos parâmetros de criação automática de topologias.

[1] http://demo.spear.narmox.com/app/?apiurl=demo#!/mininet

- "tree": Constrói uma topologia em forma de árvore como já foi explicado pelo comando 4 anteriormente nesta seção. Os *hosts* dessa topologia são equivalentes as folhas, e os galhos serão equivalentes a *switches*.
- "linear": Neste tipo de topologia, os *switches* estarão dispostos sequencialmente com a quantidade de *hosts* informado no parâmetro ligados a cada um deles.
- "single": Neste caso a construção é apenas de 1 *switch* com a quantidade de *hosts* passados pelo parâmetro ligados a ele.

A figura 12 representa uma topologia que foi criada utilizando o parâmetro --topo=linear,3,2". O número 3 neste parâmetro representa a quantidade de *switches* seguido pela quantidade de *hosts*.

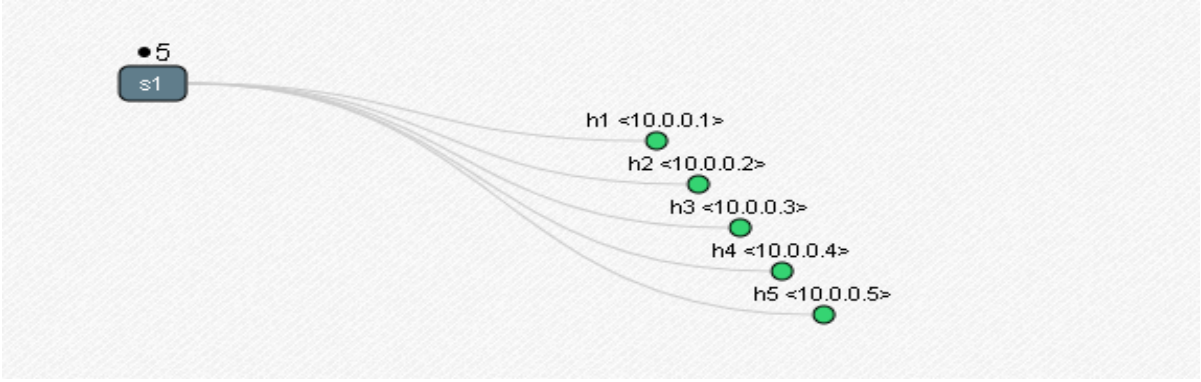
Figura 12 – Representação gráfica da topologia criada utilizando parâmetro "linear".



Fonte: Elaborado pelo Autor

Como a figura 12 mostra, a topologia foi criada com os *switches* em sequencia com seus *hosts* conectados. E por fim, a figura 13 mostra a topologia criada usando o parâmetro --topo=single,5", onde 5 representa o número de *hosts* que foram criados.

Figura 13 – Representação gráfica da topologia criada utilizando parâmetro "single".



Fonte: Elaborado pelo Autor

5.3 Criação da Topologia de Rede

O recurso de criação de topologias automáticas, conforme mencionado na seção anterior, é bastante utilizado nas simulações com o Mininet, porém esse trabalho utiliza uma topologia customizada para do cenário de uma instituição de ensino.

Neste cenário foi considerado uma topologia em árvore, onde foi definido 3 níveis de *switches*. O primeiro nível foi constituído pelo *switch* central s1. O segundo nível foi constituído por 3 *switches* s2, s3 e s4. Foi considerado para o *switch* s2 um número de 4 *hosts*, para o *switch* s3 um número total de 6 *hosts* e para o *switch* s4 como tendo outros dois *switches* ligados a ele constituindo então a terceira camada de *switches*. Na última camada com a presença dos *switches* s5 e s6 foram atribuídos a quantidade de 2 *hosts* para o primeiro e 1 *host* para o último.

O código 5 cria a topologia customizada do cenário simulado. Nas linhas 9 a 21 são criados os *hosts* (h1 a h13), nas linhas 22 a 27 definem a criação dos *switches* (s1 a s6), e finalmente, nas linhas 30 a 47 definem os links entre os *switches* e *hosts* culminando na topologia proposta.

```
1  from mininet.topo import Topo
2  class MyTopo( Topo ):
3      def __init__( self ):
4
5          # Inicializando topologia
6          Topo.__init__( self )
7
8          # Adicionando hosts e switches
9    m1h1 = self.addHost('h1')
10   m1h2 = self.addHost('h2')
11   m1h3 = self.addHost('h3')
12   m1h4 = self.addHost('h4')
13   m2h1 = self.addHost('h5')
14   m2h2 = self.addHost('h6')
15   m2h3 = self.addHost('h7')
16   m2h4 = self.addHost('h8')
17   m2h5 = self.addHost('h9')
18   m2h6 = self.addHost('h10')
19   m3h1 = self.addHost('h11')
20   m3h2 = self.addHost('h12')
21   m4h1 = self.addHost('h13')
22   mcs = self.addSwitch('s1')
23   m1s = self.addSwitch('s2')
24   m2s = self.addSwitch('s3')
25   m3s = self.addSwitch('s4')
26   m3s1 = self.addSwitch('s5')
27   m3s2 = self.addSwitch('s6')
28
29         # Adicionando links
```

```
30    self.addLink( m1h1, m1s)
31    self.addLink( m1h2, m1s)
32    self.addLink( m1h3, m1s)
33    self.addLink( m1h4, m1s)
34    self.addLink( m2h1, m2s)
35    self.addLink( m2h2, m2s)
36    self.addLink( m2h3, m2s)
37    self.addLink( m2h4, m2s)
38    self.addLink( m2h5, m2s)
39    self.addLink( m2h6, m2s)
40    self.addLink( m3h1, m3s1)
41    self.addLink( m3h2, m3s1)
42    self.addLink( m4h1, m3s2)
43    self.addLink( m1s, mcs)
44    self.addLink( m2s, mcs)
45    self.addLink( m3s1, m3s)
46    self.addLink( m3s2, m3s)
47    self.addLink( m3s, mcs)
48 topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Código 5 – Código fonte para criação de uma topologia customizada

Para a inicialização da topologia foi utilizado o seguinte comando.

```
1 Floodlight@Floodlight:~$ sudo mn --custum Desktop/customTopology.py --
    topo mytopo --mac --switch ovsk --controller=remote,ip=0.0.0.0,port
    =6653
```

Código 6 – Criação da topologia customizada no Mininet

Antes de continuar, segue uma explicação dos parâmetros usados no comando 6:

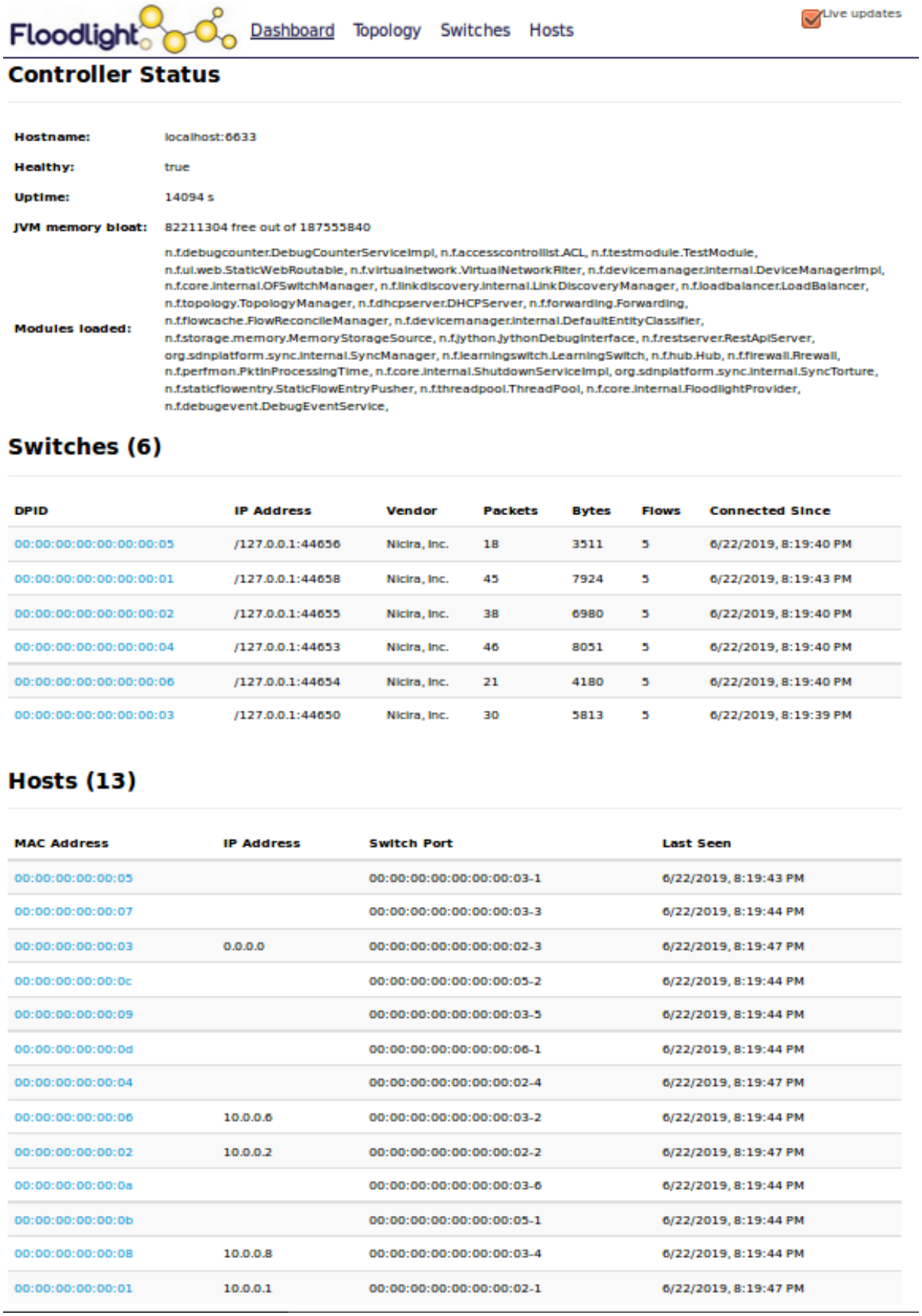
- - -custom: Indica o uso de topologia customizada.
- - -topo: Indica o tipo de topologia.
- - -mac: Faz a sequenciação dos endereços MAC dos *hosts*.
- - -switch: Especifica o tipo de switch.
- - -controller=remote,ip=[IP],port=[PORTA]: Realiza a conexão do Mininet indicando o IP e a porta do controlador remoto.

A execução do código 6 com a adição do controlador Floodlight remoto gerou uma página web[2] que ao ser acessada dispõe de informações da topologia customizada que foi criada. A figura 14 representa a página do controlador Floodlight.

Ainda na página web do controlador Floodlight, na aba *topology*, é possível a visualização gráfica da topologia criada como mostra a figura 15, resultado obtido com a execução do código 5 possibilitando visualizar os dispositivos da rede. Na topologia os *switches*

[2] http://localhost:8080/ui/index.html

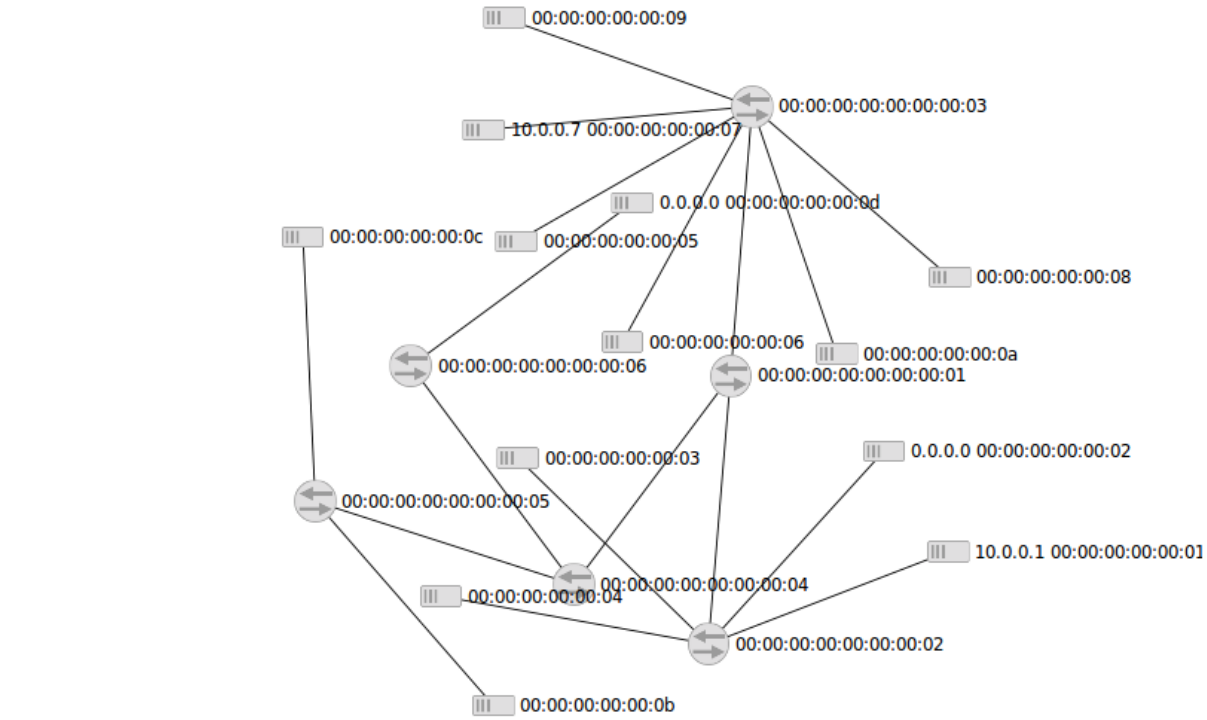
Figura 14 – Página web do controlador Floodlight.



Fonte: Elaborado pelo Autor

s2 (00:00:00:00:00:00:00:02), s3 (00:00:00:00:00:00:00:03) e s4 (00:00:00:00:00:00:00:04) conectados ao *switch* central s1 (00:00:00:00:00:00:00:01).

Figura 15 – Resultado visual da topologia customizada.



Fonte: Elaborado pelo Autor

Para verificação da comunicação entre os *hosts* pela rede, o Mininet dispõe do comando "pingall", esse comando que testa a conectividade da rede usando o protocolo ICMP entre os *hosts* e pode ser visualizada na figura 16.

Figura 16 – Teste de conectividade da topologia customizada.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12 h13
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12 h13
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12 h13
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12 h13
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12 h13
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12 h13
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12 h13
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12 h13
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h13
h13 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
*** Results: 0% dropped (156/156 received)
```

Fonte: Elaborado pelo Autor

O resultado de 0% de pacotes perdidos obtido no teste de conectividade, ilustrados na figura 16, considera um cenário sem nenhuma regra de *firewall* aplicada pelo controlador aos *switches*, possibilitando conexões livres entre todos os *hosts*.

5.4 Execução do firewall

Nas seções anteriores foram discutida a operação do cenário em que o controlador estava presente mas sem a execução do módulo do *firewall*, consequentemente sem regras de controle do tráfego. Nesta seção será discutida a configuração do módulo do *firewall* no controlador Floodlight, porém sem regras ativas aplicadas ao cenário.

O módulo do *firewall* do controlador Floodlight funciona através de uma *Application Programming Interface* (API), sendo necessário a instalação da ferramenta *curl*. A execução do código 7 possibilita a instalação da ferramenta Curl.

```
root@Floodlight:/home/Floodlight# sudo apt-get install curl
```

Código 7 – Código para instalação do Curl

A ferramenta Curl possibilita a requisição do módulo do *firewall* através do comando representado no código 8.

```
root@Floodlight:/home/Floodlight# curl http://localhost:8080/wm/firewall
    /module/enable/json -X PUT
```

Código 8 – Requisição para ativação do *firewall*

Na inicialização do módulo do *firewall* no Floodlight, o conjunto de regras carregadas bloqueiam todo o encaminhamento entre os *hosts* da rede. A validação dessa condição é feita ao ser executado novamente o comando "pingall"no Mininet. A figura 17 mostra que teve uma porcentagem de 100% de perdas de pacotes, significando que o tráfego pela rede entre os *hosts* estavam realmente bloqueados pela inicialização padrão do *firewall*.

O código 9 apresenta a regra no código fonte do módulo do *firewall* onde é definido o bloqueio de todo o tráfego da rede.

```
1     ...
2     if (matched_rule == null) {
3     Match.Builder mb = OFFactories.getFactory(pi.getVersion()).
    buildMatch();
4     mb.setExact(MatchField.IN_PORT, (pi.getVersion().compareTo(OFVersion
    .OF_12) < 0 ? pi.getInPort() : pi.getMatch().get(MatchField.IN_PORT))
    )
5     .setExact(MatchField.ETH_SRC, eth.getSourceMACAddress())
6     .setExact(MatchField.ETH_DST, eth.getDestinationMACAddress())
7     .setExact(MatchField.ETH_TYPE, eth.getEtherType());
8
9     if (mb.get(MatchField.ETH_TYPE).equals(EthType.IPv4)) {
10      IPv4 ipv4 = (IPv4) eth.getPayload();
11      mb.setExact(MatchField.IPV4_SRC, ipv4.getSourceAddress())
```

Figura 17 – Teste de conectividade da topologia customizada com o *firewall* em funcionamento

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X X X X X X X X X
h2 -> X X X X X X X X X X X X
h3 -> X X X X X X X X X X X X
h4 -> X X X X X X X X X X X X
h5 -> X X X X X X X X X X X X
h6 -> X X X X X X X X X X X X
h7 -> X X X X X X X X X X X X
h8 -> X X X X X X X X X X X X
h9 -> X X X X X X X X X X X X
h10 -> X X X X X X X X X X X X
h11 -> X X X X X X X X X X X X
h12 -> X X X X X X X X X X X X
h13 -> X X X X X X X X X X X X
*** Results: 100% dropped (0/156 received)
```

Fonte: Elaborado pelo Autor

```
12        .setExact(MatchField.IPV4_DST, ipv4.getDestinationAddress())
13        .setExact(MatchField.IP_PROTO, ipv4.getProtocol());
14
15        if (mb.get(MatchField.IP_PROTO).equals(IpProtocol.TCP)) {
16          TCP tcp = (TCP) ipv4.getPayload();
17          mb.setExact(MatchField.TCP_SRC, tcp.getSourcePort())
18          .setExact(MatchField.TCP_DST, tcp.getDestinationPort());
19        } else if (mb.get(MatchField.IP_PROTO).equals(IpProtocol.UDP)) {
20          UDP udp = (UDP) ipv4.getPayload();
21          mb.setExact(MatchField.UDP_SRC, udp.getSourcePort())
22          .setExact(MatchField.UDP_DST, udp.getDestinationPort());
23        } else {
24          // could be ICMP, which will be taken care of via IPv4 src/dst +
      ip proto
25        }
26      }
27      rmp.match = mb.build();
28      ...
```

Código 9 – Código fonte específico onde é barrado todo o tráfego na inicialização do *firewall*

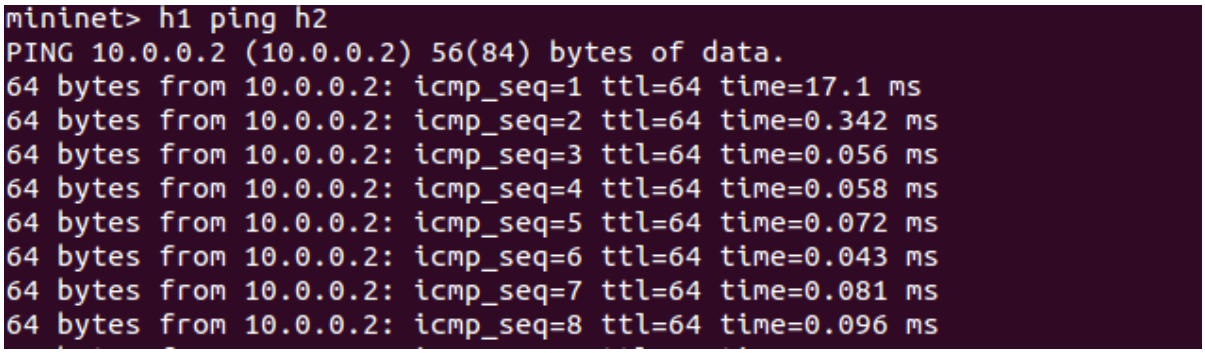
As liberações de tráfego são adicionadas ao código do fonte do *firewall* ou através do código na linha de comando. Por exemplo, o código 10 realiza a liberação de tráfego para permitir a comunicação entre todos os *hosts* do *switch* s2.

```
1 curl -X POST -d '{"switchid": "00:00:00:00:00:00:00:02"}' http://
    localhost:8080/wm/firewall/rules/json
```

Código 10 – Requisição para permitir tráfegos no *switch* s2

A figura 18 mostra o resultado obtido de um teste de conectividade entre dois entre os *hosts* h1 e o *host* h2, sendo ambos pertencentes ao *switch* s2.

Figura 18 – Teste de conectividade entre os *hosts* h1 e h2 com *firewall* funcionando.



Fonte: Elaborado pelo Autor

6 Resultados e Discussões

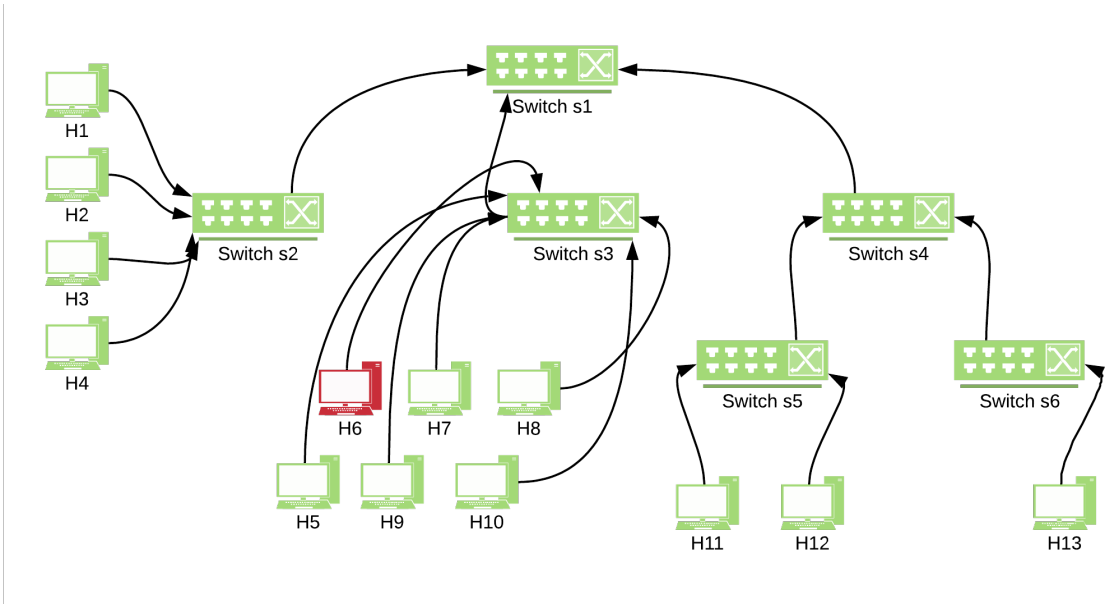
Este capítulo apresenta as regras de bloqueios e os resultados obtidos nas simulações do cenário. A seção 6.1 apresenta os resultados da simulação de bloqueio por endereço MAC. Já a seção 6.2 é apresenta o método de bloqueio por endereço IP e a Seção 6.3 simula o bloqueio por portas da camada de transporte. Finalmente a seção 6.4 apresenta uma avaliação do *firewall* do implementado usando o Floodlight.

6.1 Bloqueio por MAC

A realização dos bloqueios nos cenário simulados foram realizados considerando o contexto de uma instituição de ensino. Simulando casos de bloqueios de acessos de computadores restritos, o bloqueio de endereços IPs vulneráveis e finalmente a realização de restrições de portas da camada de aplicação.

A figura 19 apresenta o *layout* do cenário simulado considerando a regra de bloqueio de MAC adicionada ao controlador, destacando o *host* h6 em vermelho, representando a sua condição de bloqueio.

Figura 19 – Layout atual com a regra 11 em execução no controlador.



Fonte: Elaborado pelo Autor

Nessa simulação o *host* h6 contém dados críticos para o restante dos *hosts* da rede, devendo se tornar inacessível para o restante da rede. O código 11 possibilita adicionar a regra de bloqueio especificando ao módulo do *firewall*, o bloqueio de MAC do *host* h6.

```
1 curl -X POST -d '{"dst-mac": "00:00:00:00:00:06","action":"deny"}' http:
    //localhost:8080/wm/firewall/rules/json
```

```
2 curl -X POST -d '{"src-mac": "00:00:00:00:00:06","action":"deny"}' http:
    //localhost:8080/wm/firewall/rules/json
```

Código 11 – Comando para bloquear o MAC do *host* h6

A tabela 6 a seguir representa resumidamente a regra de bloqueio por MAC que o código 11 adicionou ao controlador Floodlight.

Tabela 6 – Regras para bloqueio de MAC

MAC Origem	MAC destino	Ação
*	00:00:00:00:00:06	Deny
00:00:00:00:00:06	*	Deny

Fonte: Elaborado pelo autor

Após a execução do código 11, os testes foram realizados no Mininet, executando o comando *ping* entre os *hosts* h1 e h6 e entre os *hosts* h10 e h6. A figura 20 apresenta os resultados das tentativas de comunicação dos entre os *hosts* com a regra de bloqueio de MAC em execução nos *switches*.

Figura 20 – Resultados da regra de MAC adicionada ao módulo do *firewall*

```
mininet> h1 ping h6
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=10 Destination Host Unreachable
From 10.0.0.1 icmp_seq=11 Destination Host Unreachable
From 10.0.0.1 icmp_seq=12 Destination Host Unreachable
^C
--- 10.0.0.6 ping statistics ---
14 packets transmitted, 0 received, +3 errors, 100% packet loss, time 13005ms
pipe 3
mininet> h10 ping h6
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data.
From 10.0.0.10 icmp_seq=10 Destination Host Unreachable
From 10.0.0.10 icmp_seq=11 Destination Host Unreachable
From 10.0.0.10 icmp_seq=12 Destination Host Unreachable
^C
--- 10.0.0.6 ping statistics ---
14 packets transmitted, 0 received, +3 errors, 100% packet loss, time 13005ms
pipe 3
```

(a)

```
mininet> h1 ping h7
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data.
64 bytes from 10.0.0.7: icmp_seq=1 ttl=64 time=162 ms
64 bytes from 10.0.0.7: icmp_seq=2 ttl=64 time=0.264 ms
64 bytes from 10.0.0.7: icmp_seq=3 ttl=64 time=0.055 ms
64 bytes from 10.0.0.7: icmp_seq=4 ttl=64 time=0.075 ms
64 bytes from 10.0.0.7: icmp_seq=5 ttl=64 time=0.091 ms
64 bytes from 10.0.0.7: icmp_seq=6 ttl=64 time=0.069 ms
64 bytes from 10.0.0.7: icmp_seq=7 ttl=64 time=0.076 ms
64 bytes from 10.0.0.7: icmp_seq=8 ttl=64 time=0.078 ms
64 bytes from 10.0.0.7: icmp_seq=9 ttl=64 time=0.077 ms
64 bytes from 10.0.0.7: icmp_seq=10 ttl=64 time=0.087 ms
64 bytes from 10.0.0.7: icmp_seq=11 ttl=64 time=0.079 ms
64 bytes from 10.0.0.7: icmp_seq=12 ttl=64 time=0.080 ms
64 bytes from 10.0.0.7: icmp_seq=13 ttl=64 time=0.072 ms
64 bytes from 10.0.0.7: icmp_seq=14 ttl=64 time=0.066 ms
64 bytes from 10.0.0.7: icmp_seq=15 ttl=64 time=0.072 ms
64 bytes from 10.0.0.7: icmp_seq=16 ttl=64 time=0.072 ms
```

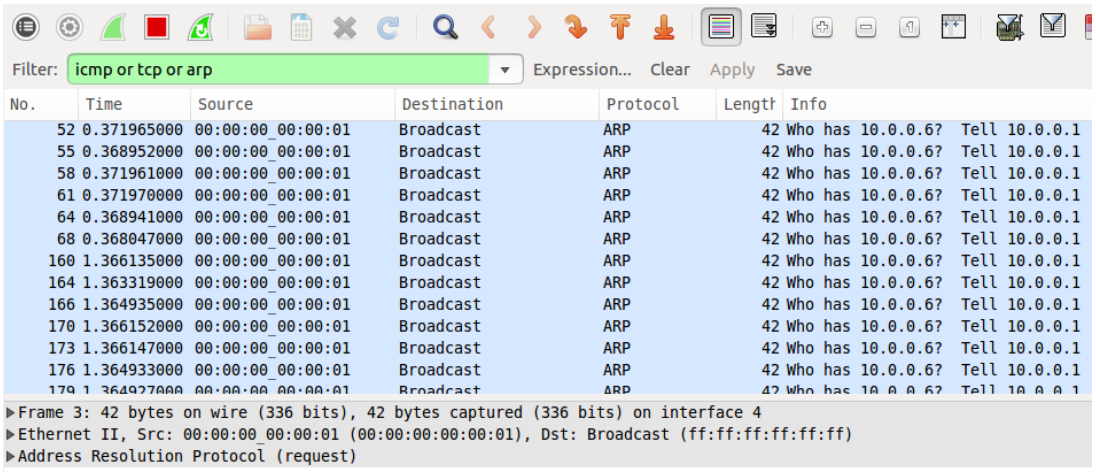
(b)

Fonte: Elaborado pelo Autor

Como pode ser visto pela parta (a) na figura 20, as tentativas de conexão com o *host* h6 pelos *hosts* h1 e h10 resultaram em 100% de perda de pacotes, comprovando o bloqueio do host. A parte (b) da figura 20 representa a comunicação entre dois *hosts* quaisquer enquanto a regra 11 está carregada nos *switches*.

A figura 21 apresenta parte do tráfego entre os hosts durante a execução do comando ping, através da ferramenta *Wireshark*, demonstrando que o MAC bloqueado está indisponível na topologia simulada.

Figura 21 – análise de tráfego entre os *hosts* h1 e h6 pelo *wireshark*.

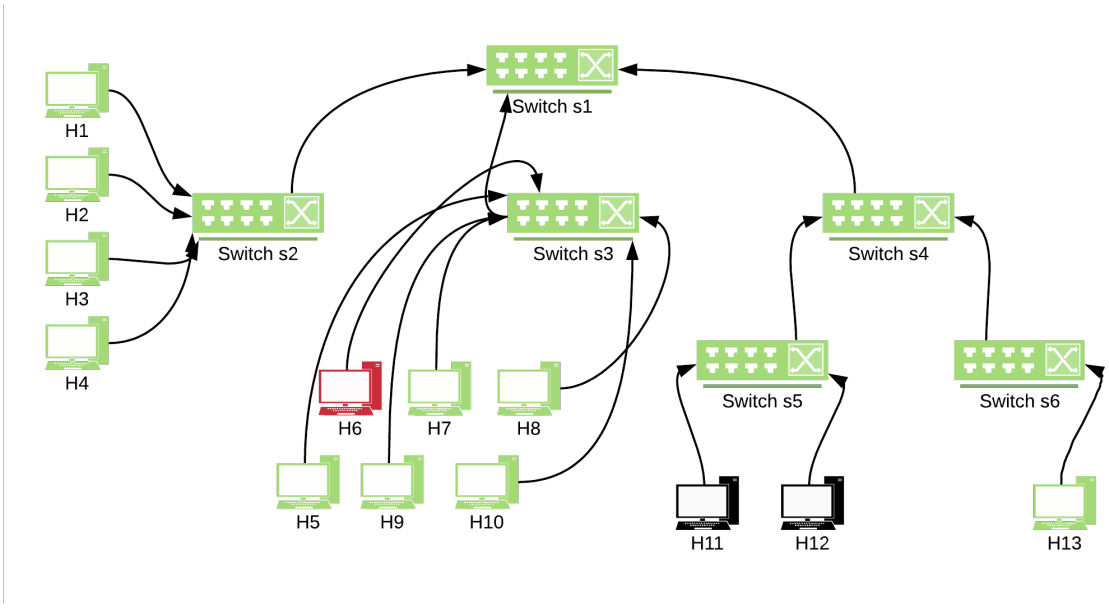


Fonte: Elaborado pelo Autor

6.2 Bloqueio por IP

A figura 22, apresenta o *layout* do cenário simulado considerando da regra de bloqueio por endereço MAC adicionando uma nova regra de bloqueio de endereços IPs ao controlador. Os *hosts* em negrito representam os *hosts* bloqueados através das regras de endereço IP, ou seja, os *hosts* h11 e h12 tiveram os seus endereços IP bloqueados.

Figura 22 – *Layout* após adição da regra de bloqueio por IP



Fonte: Elaborado pelo Autor

Ambos os *hosts* h11 e h12 foram liberados para uso de quaisquer pessoas na instituição. Em termos de segurança, essa liberdade de uso traz sérios riscos para a segurança do restante da rede. O código 12 foi executado no Mininet, para a inserir a regra de bloqueio ao controlador Floodlight, bloqueando o seu IP para o resto dos *hosts* da rede.

```
curl -X POST -d '{"src-ip": "*", "dst-ip": "10.0.0.11/32", "action": "
    deny"}' http://localhost:8080/wm/firewall/rules/json
curl -X POST -d '{"src-ip": "*", "dst-ip": "10.0.0.12/32", "action": "
    deny"}' http://localhost:8080/wm/firewall/rules/json
curl -X POST -d '{"src-ip": "10.0.0.11/32", "dst-ip": "*", "action": "
    deny"}' http://localhost:8080/wm/firewall/rules/json
curl -X POST -d '{"src-ip": "10.0.0.12/32", "dst-ip": "*", "action": "
    deny"}' http://localhost:8080/wm/firewall/rules/json
```

Código 12 – Comando para bloquear o IP dos *host* h11 e h12 da topologia

A tabela 8, resume a regra 12 adicionada ao controlador.

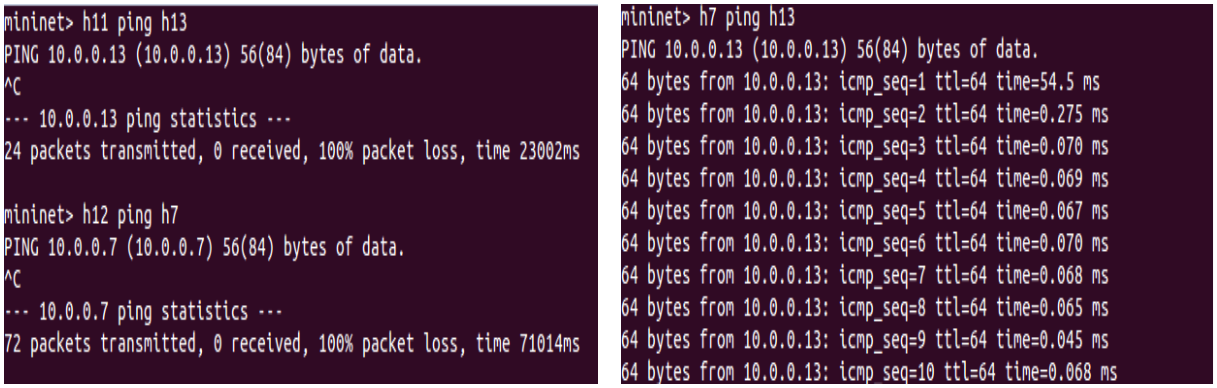
Tabela 7 – Regras para bloqueio por IP

IP Origem	IP destino	Ação
*	10.0.0.11	Deny
*	10.0.0.12	Deny
10.0.0.11	*	Deny
10.0.0.12	*	Deny

Fonte: Elaborado pelo autor

Executado o código 12, foram realizado testes de *ping* entre os *hosts* h11 e h13 e h12 com o h7. O resultado do comando pode ser observado no lado (a) da figura 23, após as simulações de conexão no Mininet. O lado (b) da figura 23 revela um caso de *ping* qualquer apresentando o comportamento dos *switches* apresentando uma conexão válida enquanto as regras de IP e MAC estão carregadas nos *switches*.

Figura 23 – Resultados da regra de IP adicionada ao módulo do *firewall*

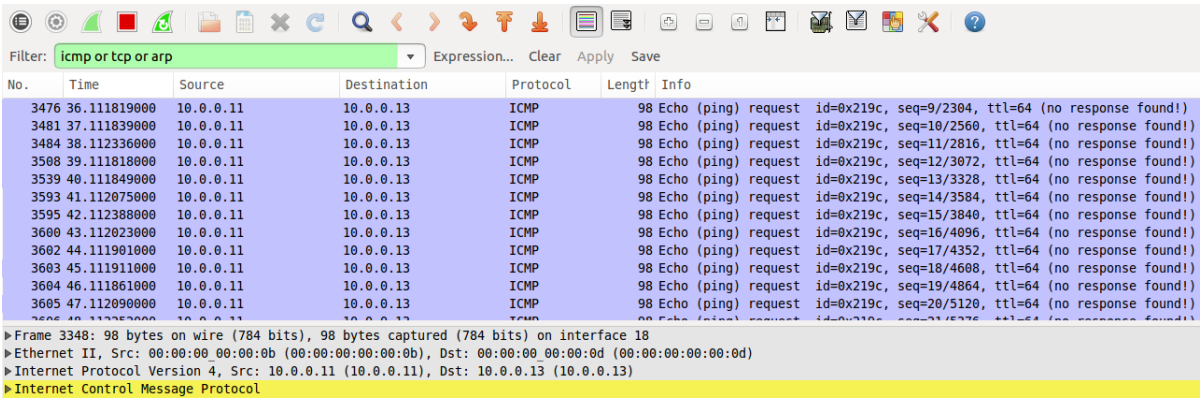


(a) (b)

Fonte: Elaborado pelo Autor

Utilizando-se da ferramenta *wireshark*, é apresentado na figura 24 a análise do tráfego entre os *hosts* h11 e 13, demonstrando que não eram obtidas respostas na tentativa de conexão que envolvesse os *hosts* com IP bloqueados.

Figura 24 – análise do tráfego entre os *hosts* h11 e h13.

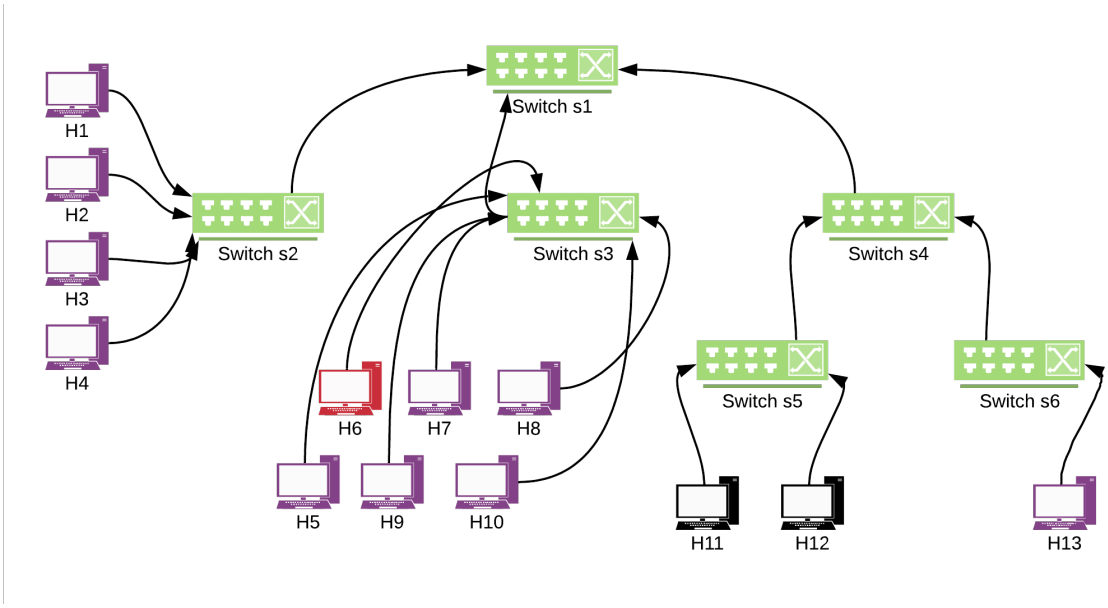


Fonte: Elaborado pelo Autor

6.3 Bloqueio por Porta TCP

Por fim, na figura 25 é apresentado o cenário da simulação considerando a regra de bloqueio de uma porta TCP adicionada ao controlador. Nesta figura, todos os *hosts* em roxo, juntamente com os *hosts* que já representam uma regra em atividade, passaram a ter uma de suas portas da camada de transporte bloqueadas.

Figura 25 – Layout da topologia com adição da regra de bloqueio por portas TCP



Fonte: Elaborado pelo Autor

A porta TCP 23 corresponde ao Telnet. O Telnet foi um protocolo de acesso remoto muito usado durante a década de 1980 e 1990, acontece que tal protocolo caiu em desuso sendo substituído pelo *Secure Shell* (SSH). A desvantagem do Telnet era o fato de ser um protocolo aberto que transmite comandos em texto simples, o que incluía logins e senhas. Esta desvantagem permitia com que o servidor fosse facilmente invadido por pessoas com más intenções. Portanto, devido a questões imprescindíveis de segurança, foi realizado o bloqueio

da porta TCP 23 em todos os *hosts*.

O código 13 representa o comando que adiciona a regra de bloqueio ao módulo do *firewall* do controlador.

```
curl -X POST -d '{"src-ip": "*", "dst-ip": "*", "nw-proto":"TCP", "tp-
    dst":"23", "action":"DENY" }' http://localhost:8080/wm/firewall/rules
    /json
```

Código 13 – Regra para o bloqueio de porta TCP

Na tabela 8 a seguir, é resumida a regra 13 adicionada ao controlador.

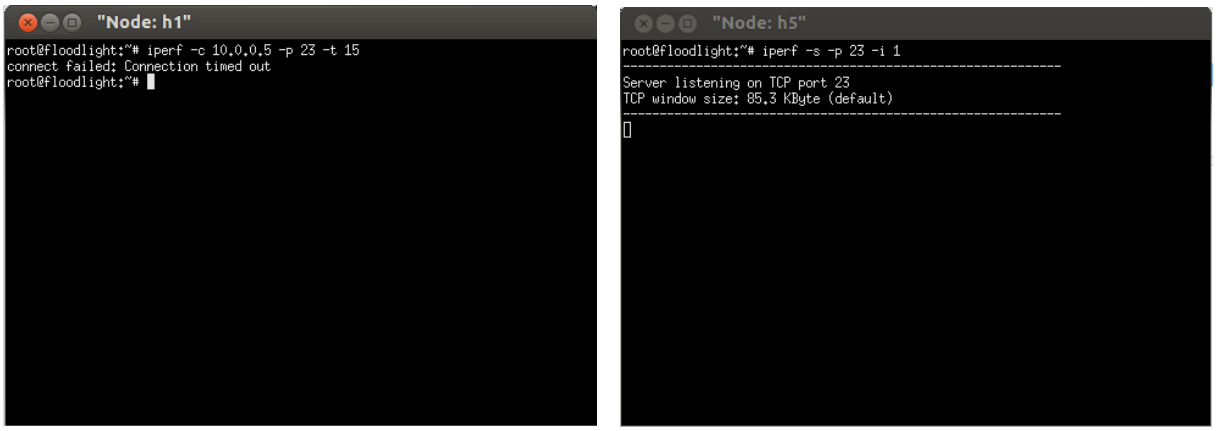
Tabela 8 – Regras para bloqueio por IP

IP Origem	IP destino	Porta	Ação
*	*	23	Deny

Fonte: Elaborado pelo autor

Feito a execução do código 13 foi executado o teste de transferência entre os *hosts* h1 e h5 pela porta TCP 23. A figura 26 mostra o resultado após o controlador ter carregado a regra de bloqueio nos *switches*.

Figura 26 – Resultados da regra de TCP adicionada ao módulo do *firewall*



(a) (b)

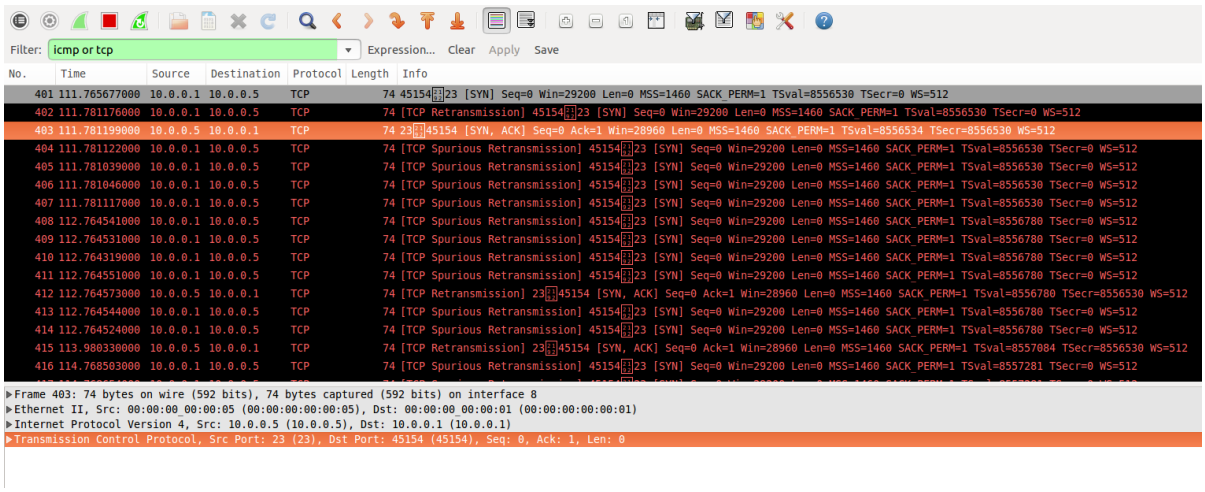
Fonte: Elaborado pelo Autor

O lado (a) da figura 26 apresenta o *host* h1 como cliente, exibindo a mensagem *"connect failed: Connection timed out"* após a tentativa de se conectar ao *host* servidor h5, representado pelo lado (b) da figura 26. A mensagem no *host* h1 demonstra que o *switch* barrou com sucesso qualquer tentativa de tráfego de dados na porta bloqueada.

O tráfego ocorrido durante essa tentativa de transferência pode ser visualizada pela figura 27, onde as tentativas de conexão tiveram a mensagem *"TCP retransmissions"*. O retorno dessa mensagem é diretamente relacionado ao princípio de entrega confiável do protocolo TCP, significando que os segmentos transmitidos não foram completados demonstrando

assim o bloqueio eficiente de qualquer transferência na porta barrada pela regra adicionada ao módulo do *firewall* pelos *switches*.

Figura 27 – análise de tráfego pelo *wireshark* durante transmissão TCP contendo a regra de bloqueio



Fonte: Elaborado pelo Autor

6.4 Discussões

A tabela 9 mostra as regras de bloqueios que foram usadas em todos os tópicos anteriores de forma resumida.

Tabela 9 – Apresentação das regras de bloqueio utilizadas na obtenção dos resultados

MAC Origem	MAC Destino	IP Origem	IP Destino	Porta TCP	Ação
*	00:00:00:00:00:06	-	-	-	Deny
-	-	*	10.0.0.11	-	Deny
-	-	*	10.0.0.12	-	Deny
-	-	10.0.0.11	*	-	Deny
-	-	10.0.0.12	*	-	Deny
-	-	*	*	23	Deny

Fonte: Elaborado pelo Autor

Os dados da tabela 9 foram dispostos de forma a resumir as regras usadas nas simulações. Todas as regras acimas foram feitas utilizando comandos no Mininet. No contexto de execução do controlador floodlight o código 14 representa o código fonte específico do *firewall* proposto, onde são feitas as permissões e bloqueios de todos os métodos envolvidos na tabela 9.

```
1 protected RuleMatchPair matchWithRule(IOFSwitch sw, OFPacketIn pi,
    FloodlightContext cntx) {
2     FirewallRule matched_rule = null;
3     Ethernet eth = IFloodlightProviderService.bcStore.get(cntx,
    IFloodlightProviderService.CONTEXT_PI_PAYLOAD);
```

```
4       AllowDropPair adp = new AllowDropPair(sw.getOFFactory());
5
6       synchronized (rules) {
7         Iterator<FirewallRule> iter = this.rules.iterator();
8         FirewallRule rule = null;
9         // iterate through list to find a matching firewall rule
10        while (iter.hasNext()) {
11          // get next rule from list
12          rule = iter.next();
13
14          // check if rule matches
15          // AllowDropPair adp's allow and drop matches will modified with
        what matches
16          if (rule.matchesThisPacket(sw.getId(), (pi.getVersion().
      compareTo(OFVersion.OF_12) < 0 ? pi.getInPort() : pi.getMatch().get(
      MatchField.IN_PORT)), eth, adp) == true) {
17            matched_rule = rule;
18            break;
19          }
20        }
21      }
22
23      // make a pair of rule and wildcards, then return it
24      RuleMatchPair rmp = new RuleMatchPair();
25      rmp.rule = matched_rule;
26      if (matched_rule == null) {
27        /*
28         * No rule was found, so drop the packet with as specific
29         * of a drop rule as possible as not to interfere with other
30         * firewall rules.
31         */
32        Match.Builder mb = OFFactories.getFactory(pi.getVersion()).
      buildMatch();
33        mb.setExact(MatchField.IN_PORT, (pi.getVersion().compareTo(
      OFVersion.OF_12) < 0 ? pi.getInPort() : pi.getMatch().get(MatchField.
      IN_PORT)))
34        .setExact(MatchField.ETH_SRC, eth.getSourceMACAddress())
35        .setExact(MatchField.ETH_DST, eth.getDestinationMACAddress())
36        .setExact(MatchField.ETH_TYPE, eth.getEtherType());
37
38        if (mb.get(MatchField.ETH_TYPE).equals(EthType.IPv4)) {
39          IPv4 ipv4 = (IPv4) eth.getPayload();
40          mb.setExact(MatchField.IPV4_SRC, ipv4.getSourceAddress())
41          .setExact(MatchField.IPV4_DST, ipv4.getDestinationAddress())
42          .setExact(MatchField.IP_PROTO, ipv4.getProtocol());
43
44          if (mb.get(MatchField.IP_PROTO).equals(IpProtocol.TCP)) {
45            TCP tcp = (TCP) ipv4.getPayload();
```

```
46            mb.setExact(MatchField.TCP_SRC, tcp.getSourcePort())
47            .setExact(MatchField.TCP_DST, tcp.getDestinationPort());
48          } else if (mb.get(MatchField.IP_PROTO).equals(IpProtocol.UDP)) {
49            UDP udp = (UDP) ipv4.getPayload();
50            mb.setExact(MatchField.UDP_SRC, udp.getSourcePort())
51            .setExact(MatchField.UDP_DST, udp.getDestinationPort());
52          } else {
53            // could be ICMP, which will be taken care of via IPv4 src/dst
    + ip proto
54          }
55        }
56        rmp.match = mb.build();
57        //rmp.match = adp.drop.build(); This inserted a "drop all" rule if
    no match was found (not what we want to do...)
58      } else if (matched_rule.action == FirewallRule.FirewallAction.DROP)
    {
59        rmp.match = adp.drop.build();
60      } else {
61        rmp.match = adp.allow.build();
62      }
63      return rmp;
64    }
```

Código 14 – Código fonte do *firewall* onde são efetivadas as regras de tráfego na topologia simulada

Observando o trecho de código fonte 14, na linha 59 é possível realizar os comandos para negar um pacote, de acordo com a definição das regras. Já na linha 61 encontra-se o comando que permite o tráfego de um pacote, conforme definição do administrador.

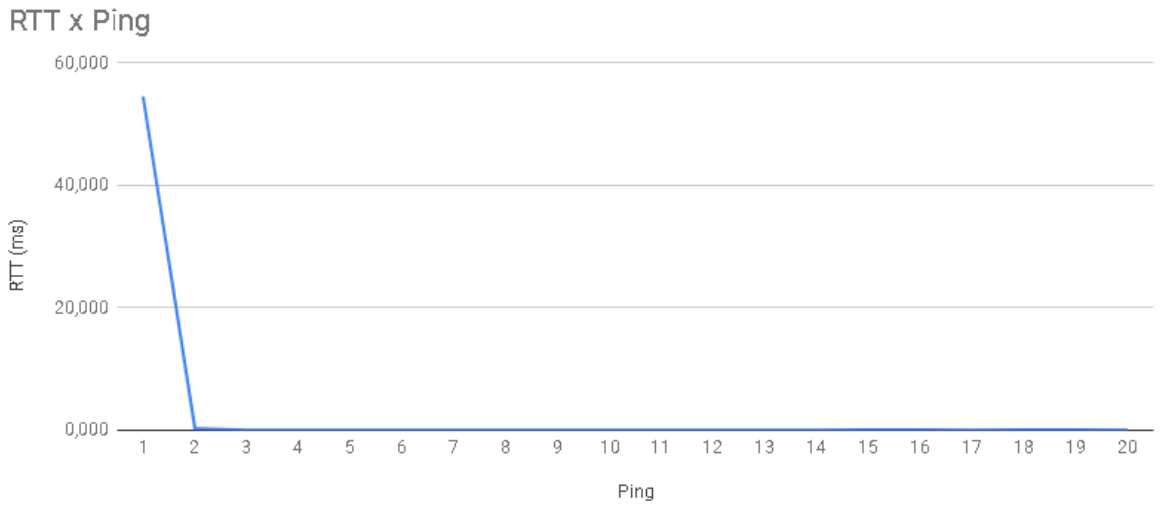
6.4.1 Tempos de Round time trip obtidos

O *Round-Trip time* (RTT) no ambiente de redes é tido como o tempo total que uma solicitação percorre do *host* inicial até o *host* final adicionado ao tempo de retorno da confirmação de sucesso da solicitação. Uma importante observação realizada sobre as figuras 20 e 23 foi em relação aos tempos de RTT exibidos durante a realização bem sucedida do ping entre os *hosts* de testes, ambos presentes ao lado (b) das figuras. Pensando nesses tempos, foram selecionados os tempos das 20 primeiras solicitações de um ping geral entre *hosts* com as regras carregadas nos *switches* e dispostos em um gráfico que pode ser observado pela figura 28.

Na figura 28, foi observada uma abrupta queda no RTT em relação ao primeiro pacote da solicitação e o restante dos pacotes. Essa variação no tempo de reposta ocorre devido a consulta realizada entre os *switches* e o controlador, sendo necessário carregar as regras e as definições nos *switches*. Após a consulta, o tempo de reposta diminui drasticamente, pois após o carregamento das regras nos *switches* não são realizadas novas consultas.

Portanto, uma vez que as regras de encaminhamento são carregadas nos *switches*,

Figura 28 – Gráfico representando o RTT do ping entre *hosts* na topologia com regras de bloqueio em ação



Fonte: Elaborado pelo Autor

não são necessárias novas consultas ao controlador, de forma que as definições de encaminhamento posteriores ocorrem nos *switches*, consequentemente, reduzindo o tempo de latência.

7 Conclusão

O trabalho apresentou uma opção para as regras de seguranças de uma rede com o uso do paradigma de SDN. Ao propor o uso do *firewall* utilizando o controlador Floodlight no cenário simulando a rede de uma instituição de ensino, possibilitou definir as regras de tráfego de rede de maneira centralizada e flexível, usando o console de gerenciamento do controlador de forma a melhorar o gerenciamento do conteúdo como um todo.

De acordo com as simulações o *firewall* SDN em relação ao modelo de *firewall* tradicional, permite ao administrador da rede a manipulação direta no código fonte para implementação ou atualização de políticas de encaminhamento de acordo com a necessidade do cenário e do ambiente a ser aplicado.

Conclui-se também que a proposta do *firewall* como aplicação na rede agrega ao cenário algumas vantagens em relação aos sistemas legados. A possibilidade de execução dos filtros com base nas regras serem realizadas nos dispositivos de encaminhamento e o conhecimento das regras de encaminhamento sem a constante consulta ao controlador são as principais vantagens destacadas na realização do trabalho.

A principal limitação acerca do desenvolvimento desse trabalho se deu em respeito a disponibilidade de equipamentos compatíveis com o protocolo OpenFlow na instituição de ensino ao qual foi pensado este trabalho, sendo identificado apenas 3 *switches* compatíveis, consequentemente optando-se pela simulação como abordagem para este trabalho.

A principal sugestão para um trabalho futuro fica a implementação dessa proposta em um cenário real com a utilização de equipamentos compatíveis com o OpenFlow. Seguindo no contexto de segurança de redes, fica a opção de implementação de ferramentas de segurança como o IPS e IDS. O potencial do paradigma SDN também pode ser explorado em trabalhos futuros, seja no estudo sobre o Openflow ou sobre o uso de múltiplos controladores SDN. A exploração do tema está aberta a novos trabalhos e são muitas as possibilidades que o paradigma SDN pode proporcionar atualmente no campo de pesquisa de redes de computadores.

Referências

BELLOVIN, S. M. *Distributed firewalls*. [S.l.]: login, 1999. Citado na página 14.

BELLOVIN, S. M.; CHESWICK, W. R. Network firewalls. *IEEE communications magazine*, IEEE, v. 32, n. 9, p. 50–57, 1994. Citado na página 24.

BHOLEBAWA, I. Z.; DALAL, U. D. Performance analysis of sdn/openflow controllers: Pox versus floodlight. *Wireless Personal Communications*, Springer, v. 98, n. 2, p. 1679–1699, 2018. Citado na página 33.

BHOLEBAWA, I. Z.; JHA, R. K.; DALAL, U. D. Performance analysis of proposed openflow-based network architecture using mininet. *Wireless Personal Communications*, v. 86, n. 2, p. 943–958, Jan 2016. ISSN 1572-834X. Disponível em: <https://doi.org/10.1007/s11277-015-2963-4>. Citado na página 30.

BOUCADAIR, M.; JACQUENET, C. *Software-defined networking: A perspective from within a service provider environment*. [S.l.], 2014. Citado na página 14.

CHOWDHARY, A. et al. Sdfw: Sdn-based stateful distributed firewall. *arXiv preprint arXiv:1811.00634*, 2018. Citado nas páginas 8, 24, 35 e 36.

COMER, D. *Interligação de Redes com TCP/IP–: Princípios, Protocolos e Arquitetura*. [S.l.]: Elsevier Brasil, 2016. v. 1. Citado nas páginas 19, 20, 21, 25, 26, 29 e 30.

COMER, D. E. *Redes de Computadores e Internet-6*. [S.l.]: Bookman Editora, 2016. Citado nas páginas 16, 17 e 32.

FOROUZAN, B. A. *Comunicação de dados e redes de computadores*. [S.l.]: AMGH Editora, 2009. Citado nas páginas 17 e 18.

FOROUZAN, B. A.; FEGAN, S. C. *Protocolo TCP/IP-3*. [S.l.]: AMGH Editora, 2009. Citado na página 16.

GUPTA, V.; KAUR, S.; KAUR, K. Implementation of stateful firewall using pox controller. In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. [S.l.: s.n.], 2016. p. 1093–1096. Citado na página 14.

HU, F.; HAO, Q.; BAO, K. A survey on software-defined network and openflow: From concept to implementation. *IEEE Communications Surveys & Tutorials*, IEEE, v. 16, n. 4, p. 2181–2206, 2014. Citado na página 14.

JESUS, R. M. de et al. Avaliação dos controladores beacon e floodlight, com o uso do emulador mininet aplicado a redes definidas por software–sdn evaluation of beacon and floodlight drivers, with the use of mininet emulator applied in defined netowrks software–sdn. 2016. Citado na página 41.

KAUR, K. et al. Programmable firewall using software defined networking. In: IEEE. *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on*. [S.l.], 2015. p. 2125–2129. Citado na página 15.

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, Ieee, v. 103, n. 1, p. 14–76, 2015. Citado nas páginas 14, 27 e 34.

KUROSE, J. F.; ROSS, K. W. *Computer networking: a top-down approach*. [S.l.]: Addison Wesley, 2007. Citado nas páginas 18, 19, 20, 21, 22, 23, 24 e 26.

MASOUDI, R.; GHAFFARI, A. Software defined networks: A survey. *Journal of Network and Computer Applications*, v. 67, p. 1 – 25, 2016. ISSN 1084-8045. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1084804516300297>. Citado nas páginas 14, 27, 28, 32, 33 e 39.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>. Citado nas páginas 30 e 31.

NEUPANE, K.; HADDAD, R.; CHEN, L. Next generation firewall for network security: A survey. In: *SoutheastCon 2018*. [S.l.: s.n.], 2018. p. 1–6. ISSN 1558-058X. Citado na página 23.

NUNES, B. A. A. et al. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, IEEE, v. 16, n. 3, p. 1617–1634, 2014. Citado na página 27.

Pena, J. G. V.; Yu, W. E. Development of a distributed firewall using software defined networking technology. In: *2014 4th IEEE International Conference on Information Science and Technology*. [S.l.: s.n.], 2014. p. 449–452. ISSN 2164-4357. Citado na página 36.

Pessoa, T. Q.; Duarte-Figueiredo, F. Nodepi : An integrated platform for smart homes. In: *2017 IEEE 9th Latin-American Conference on Communications (LATINCOM)*. [S.l.: s.n.], 2017. p. 1–6. Citado na página 37.

SANDHYA; SINHA, Y.; HARIBABU, K. A survey: Hybrid sdn. *Journal of Network and Computer Applications*, v. 100, p. 35 – 55, 2017. ISSN 1084-8045. Disponível em: <http://www.sciencedirect.com/science/article/pii/S108480451730317X>. Citado na página 27.

SATASIYA, D. et al. Analysis of software defined network firewall (sdf). In: IEEE. *Wireless Communications, Signal Processing and Networking (WiSPNET), International Conference on*. [S.l.], 2016. p. 228–231. Citado na página 15.

SCHULTZ, E. E. A framework for understanding and predicting insider attacks. *Computers & Security*, Elsevier, v. 21, n. 6, p. 526–531, 2002. Citado na página 14.

SHARMA, R.; PAREKH, C. Firewalls: A study and its classification. *International Journal of Advanced Research in Computer Science*, International Journal of Advanced Research in Computer Science, v. 8, n. 5, 2017. Citado na página 23.

STATS, I. L. Internet users in the world. 2016. Citado na página 23.

Suh, M. et al. Building firewall over the software-defined network controller. In: *16th International Conference on Advanced Communication Technology*. [S.l.: s.n.], 2014. p. 744–748. ISSN 1738-9445. Citado na página 36.

TANENBAUM, A. *S.; J WETHERALL, David. Redes de Computadores*. [S.l.]: São Paulo: Pearson Education Brasil, 2011. Citado nas páginas 16, 17, 18, 19, 20, 21, 22, 23, 24 e 25.

XIA, W. et al. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, IEEE, v. 17, n. 1, p. 27–51, 2015. Citado na página 30.

YAZDINEJAD, A.; BOHLOOLI, A.; JAMSHIDI, K. Efficient design and hardware implementation of the openflow v1. 3 switch on the virtex-6 fpga ml605. *The Journal of Supercomputing*, Springer, v. 74, n. 3, p. 1299–1320, 2018. Citado nas páginas 30 e 31.

ZHANG, Y. et al. A survey on software defined networking with multiple controllers. *Journal of Network and Computer Applications*, Elsevier, v. 103, p. 101–118, 2018. Citado nas páginas 28, 29, 32 e 33.