UNIVERSIDADE FEDERAL FLUMINENSE

HENRIQUE SANTOS FERNANDES

Provendo Segurança em Redes Definidas por Software Através da Integração com Sistemas de Detecção e Prevenção de Intrusão

NITERÓI

2016

UNIVERSIDADE FEDERAL FLUMINENSE

HENRIQUE SANTOS FERNANDES

Provendo Segurança em Redes Definidas por Software Através da Integração com Sistemas de Detecção e Prevenção de Intrusão

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia de Telecomunicações da Universidade Federal Fluminense como parte dos requisitos necessários à obtenção do título de Mestre Engenheiro de Telecomunicações. Área de concentração: Sistemas de Telecomunicações

Orientador:
Natalia Castro Fernandes

NITERÓI

2016

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

F363 Fernandes, Henrique Santos
Provendo segurança em redes definidas por software através da integração com sistemas de detecção e prevenção de intrusão / Henrique Santos Fernandes. – Niterói, RJ : [s.n.], 2016.
79 f.

Dissertação (Mestrado em Engenharia Elétrica e de Telecomunicações) - Universidade Federal Fluminense, 2016.
Orientador: Natalia Castro Fernandes.

1. Rede de computadores. 2. Segurança de dados on-line. 3. Fluxo contínuo. 4. Sistema de telecomunicação. I. Título.

CDD 004.6

HENRIQUE SANTOS FERNANDES

PROVENDO SEGURANÇA EM REDES DEFINIDAS POR SOFTWARE ATRAVÉS DA INTEGRAÇÃO COM SISTEMAS DE DETECÇÃO E PREVENÇÃO DE INTRUSÃO

Dissertação de Mestrado apresentada ao Programa de Pós Graduação em Engenharia Elétrica e de Telecomunicações da Universidade Federal Fluminense como requisito parcial para a Obtenção do Grau de Mestre em Engenharia Elétrica e de Telecomunicações.
Área de concentração: Sistemas de Telecomunicações

BANCA EXAMINADORA

Prof.ª Dra. Natalia Castro Fernandes - Orientadora
Universidade Federal Fluminense - UFF

Prof. Dr. Igor Monteiro Moraes
Instituto de Computação/Universidade Federal Fluminense – UFF

Prof. Dr. Joel André Ferreira dos Santos
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca/CEFET-RJ

Niterói
Agosto/2016

Dedico este trabalho aos meus amigos, principalmente aqueles amigos que reclamavam que eu enrolava no mestrado.

Henrique Santos Fernandes

Agradecimentos

Agradeço a professora Natalia Castro Fernandes por toda orientação que foi dada durante minha formação como Mestre Engenheiro de Telecomunicações.

Henrique Santos Fernandes.

Resumo

Os Sistemas de Detecção e Prevenção de Intrusão são fundamentais para a segurança da rede de computadores, inspecionar o tráfego da rede em tempo real em busca de intrusos para garantir uma rede confiável é um dos seus papéis. Porém a falta de integração com os ativos da rede é um dos principais fatores que limitam sua atuação. O conceito de Redes Definidas por *Software* visa diminuir a falta de integração entre os ativos de rede devido a separação do plano de dados do plano de controle. Diante da limitação da integração entre os ativos de redes e os Sistemas de Detecção e Prevenção de Intrusão, o presente estudo propõe, desenvolve e demonstra o IDSFlow, um modelo de integração de sistemas de detecção de intrusão em redes definidas por *software*. Para validar o IDSFlow, foram realizados testes utilizando o Openflow, o Mininet, CPqD e o Snort. Os resultados obtidos pelos algorítimos desenvolvidos e apresentados mostram a capacidade de integração proposta, é possível verificar a viabilidade de utilizar as regras já existentes e funcionais para o Snort assim como utilizar o histórico de utilização da rede para aumentar a efetividade da detecção e dos bloqueios de intrusos.

Palavras-chave: SDN, Openflow, Snort, IDS, Redes, Rede de Computadores, IPS, NIDS, Segurança, Analise de tráfego, Redes definidas por software.

Abstract

Intrusion Detection and Prevention Systems are fundamental to the network security, to inspect the traffic in real time seeking intruders to ensure a reliable network is one of it's roles. However the lack of integration between the network equipments, is one of the biggest factors to limit its operations. The concept of Software Defined Networks aims to reduce the lack of integration among network assets due to the separation of the data plan from the control plan. Given the limitation of integration between networks assets and Intrusion Detection and Prevention Systems, the present study proposes, develops and demonstrates IDSFlow, an integration model of intrusion detection systems in software-defined networks. To validate IDSFlow, tests were run using Openflow, Mininet, CPqD and Snort. The results obtained by the algorithms developed and presented show the proposed integration capacity, it is possible to verify the feasibility of using the existing and functional rules for Snort as well as to use the network usage history to increase the effectiveness of intrusion detection and block.

Keywords: SDN, Software defined networks, IPS, IDS, NIDS, Networks, Network Security, Security.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API	:	Application Programming Interface;
DDoS	:	Distributed Denial of Service;
DNS	:	Domain Name System;
DoS	:	Denial of Service;
FIB	:	Forwarding Information Base;
HIDS	:	Host Intrusion Detection Systems;
ICMP	:	Internet Control Message Protocol;
IDPS	:	Intrusion Detection and Prevention System;
IDS	:	Intrusion Detection System;
IP	:	Internet Protocol;
IPS	:	Intrusion Prevention System;
JSON	:	JavaScript Object Notation;
MPLS	:	Multiprotocol Label Switching;
NIDS	:	Network Intrusion Detection Systems;
OISF	:	Open Information Security Foundation;
ONF	:	Open Networking Foundation;
RIB	:	Routing Information Base;
SDN	:	Software-Defined Networking;
SSL	:	Secure Socket Layer;
TCP	:	Transmission Control Protocol;
UDP	:	User Datagram Protocol;
VLAN	:	Virtual Local Area Network;

Sumário

Capítulo 1

Introdução

A segurança da informação é um assunto que a cada dia se torna mais relevante para arquitetos e engenheiros de redes de dados. No momento da sua concepção, as redes de dados tinham um objetivo puramente acadêmico, com poucos e conhecidos usuários, onde era possível confiar no meio utilizado parar trafegar os dados e nos seus pares. A utilização massiva e amplamente interligada para a qual estas redes evoluíram abriu não só um imenso leque de possibilidades de crescimento e acesso à informação, mas também, com ele, a preocupação com a integridade dos sistemas e a privacidade de dados.

Redes de dados, sejam elas privadas ou públicas, estão a todo momento vulneráveis a ataques de usuários ou sistemas mal intencionados em busca de informações que possam ser trocadas ou vendidas. Prover a segurança destes ambientes é tão importante que se tornou não só um campo de pesquisa, mas também um nicho de mercado altamente aquecido. Para quantificar esta ideia, no relatório da *Symantec* [1], realizado no ano de 2015, utilizando dados do ano anterior, estão descritos diversos tipos de ataques que se valem de falhas nas muitas plataformas que atualmente podem ser interconectadas através de uma rede de dados. Como exemplo do ambiente hostil que estes sistemas estão expostos, temos o número de 496,657 ataques a sites por dia na Internet.

Além do relatório supracitado, que faz um levantamento de um ano inteiro, também são regularmente divulgados boletins que relatam brechas de segurança que foram encontradas, o que inicia o processo de correção dos softwares, juntamente com um alerta de que estes recursos devem ser atualizados pelos usuários para sua melhor segurança. Seja por ingerência ou impossibilidade técnica, a não atualização dos sistemas é um dos fatores mais predominantes para a ocorrência de um ataque.

Em grandes escalas, a gerência de cada recurso de forma independente se torna cada

vez mais complicada. Além disso, plataformas legadas, que já não possuem suporte e não podem ser atualizadas quando brechas em sua implementação são expostas, geraram a necessidade de uma forma externa de identificação destes ataques. Neste contexto, foram pensadas ferramentas que identifiquem tentativas de intrusão a estes sistemas através de padrões de comportamento de rede observados. Estas ferramentas são chamadas de Sistemas de Detecção de Intrusão de Rede (*Network Intrusion Detection Systems* – NIDSs). Uma evolução dos NIDSs foram os Sistemas de Prevenção de Intrusão (*Intrusion Prevention System* – IPS), que além de detectar, estão aptos a tomar ações automaticamente após uma detecção, visando impedir que esta brecha seja explorada.

Com a mudança de paradigma trazida pela implementação de uma rede de dados definida por software (SDN - *Software-Defined Networking*), novas possibilidades surgem para a gerência e o controle da rede. Nas SDNs, existe uma separação física entre o plano de dados e o plano de controle da rede, ou seja, acontece uma migração da inteligência que hoje existe em cada elemento da rede para um ou mais controladores. Assim, com esse novo paradigma de rede, busca-se diminuir a curva de aprendizagem e o tempo desprendido para manutenção e reconfiguração de elementos, podendo também, trazer boas práticas e todo conhecimento acumulado no desenvolvimento de softwares para a administração de redes de dados.

No entanto, com a introdução de novos elementos no ambiente das redes de dados, tem-se uma mudança brusca na arquitetura ossificada que se tem atualmente, gerando novas preocupações de segurança que se somam às preocupações já existentes [2]. Como era de se esperar, as soluções aplicadas na estrutura atual não são completamente aderentes ao novo modelo proposto pelas SDNs e precisam ser remodeladas e avaliadas.

Dentre muitos, um dos elementos críticos para segurança que mais será impactado pelo uso das SDNs são os sistemas de detecção de intrusão. Na estrutura atual, os IDSs são implementados em entroncamentos, com a adição de um novo elemento como ponte, podendo assim ter acesso a um percentual maior dos fluxos e se tornando o divisor entre áreas de maior e menor segurança de uma rede. Essa forma de implementação não é aderente a um ambiente de rede distribuída que tem como um de seus objetivos, evitar a criação destes pontos de convergência de tráfego, possibilitando um uso mais efetivo e racional dos recursos físicos disponíveis. Por exemplo, é possível utilizar um link de dados sobre fibra ótica, somente para tráfego em tempo real, como videoconferências, e pode-se usar o entroncamento de rádio para o tráfego de e-mail. Assim, não é trivial saber onde se tem um ponto onde a análise de tráfego seria efetiva, isso também se dá devido

à facilidade de alteração dos fluxos em uma SDN.

1.1 Objetivos

Neste contexto, este trabalho se debruça sobre o problema de aderência dos IDSs ao novo ambiente das SDNs. Especificamente, esse estudo se data baseado no uso da plataforma de SDN chamada OpenFlow [3]. Assim, propõe-se uma nova forma de implementação deste tipo de sistema, buscando com isso, apresentar uma solução para integrar um controlador OpenFlow, com um ou mais IPS/NIDS em uma rede fisicamente distribuída.

Por se tratar de uma tendência da área de tecnologia, este tópico tem vários desdobramentos em termos de pesquisa e desenvolvimento. Nestes, pontos como viabilidade, modelos e desafios de implementação, segurança e capacidade são avaliados. Tais pontos são abordados nessa dissertação por meio da avaliação da proposta desenvolvida.

A arquitetura proposta visa manter os pontos fortes de uma arquitetura distribuída anteriormente citados, e, com o sistema proposto, chamado de IDSFlow, realizar a integração da rede OpenFlow com o IDS. O IDSFlow leva automaticamente o tráfego de interesse até os pontos de análise, diferentemente dos esquemas atuais, no qual o IDS é colocado como uma ponte por onde já passa o trafego de interesse. Em uma rede SDN, o controlador possui conhecimento sobre todos os fluxos que por ela trafegam, o que torna possível realizar uma replicação destes fluxos para os elementos IPS/NIDS, levando a capacidade e o percentual de utilização do elemento IDS em consideração, evitando assim, causar a sobrecarga desses elementos e dos links utilizados para essa análise. Ter isto em mente é necessário, pois não é possível enviar apenas um percentual de pacotes de cada um dos fluxos para o IPS/NIDS quando falamos de redes OpenFlow.

A partir dos fluxos recebidos, os quais contém dentre outros elementos de controle, o pacote original, é possível realizar uma análise do conteúdo e aplicar sobre ele as técnicas de identificação que já existem e se provaram eficazes nas redes atuais. Assim, o IDSFlow pode se valer da sua integração com IDS, como fonte de informação e a partir dela, adaptar as regras já existentes ou gerar novas regras para os fluxos, que junto ao controlador OpenFlow serão aplicadas a rede sobre sua gerência. Assim, o IDSFlow trabalha na coleta, identificação, tomada de decisão e aplicação de ações para o bloqueio de tráfegos maliciosos.

Para analisar a viabilidade da proposta, foram elaborados testes que, visam validar a sua aplicabilidade. Também foram feitos testes de capacidade, para identificar problemas críticos nos principais elementos da solução que poderiam inviabilizar sua aplicação em uma rede com dados reais e, com estes resultados, passa-se a simulação de modelos de coleta e identificação de tráfego de interesse, por fim são utilizados modelos de alteração dos fluxos para diferentes casos.

Com os dados obtidos nos testes, é possível sanar as dúvidas levantadas sobre a viabilidade da proposta e também tirar lições em decorrência de obstáculos encontrados durante a execução que poderão servir de balizadores para uma implementação real do sistema proposto.

1.2 Organização do trabalho

O restante do trabalho está organizado da seguinte forma. No Capítulo 2 são mostradas soluções de segurança de redes. Já no Capítulo 3 é apresentado a SDN e sua implementação. No Capítulo 4, trabalhos relacionados são descritos. No Capítulo 5, a proposta do trabalho é introduzida, enquanto que, no Capítulo 6, os experimentos usados para analisar a proposta e os resultados obtidos são discutidos. Por fim, no Capítulo 7, as conclusões e trabalhos futuros são apresentados.

Capítulo 2

Segurança em Redes

As redes de computadores cresceram a partir de um projeto com o objetivo de acadêmico, onde o fundamental era estabelecer um meio de interatividade entre computadores fisicamente distantes. Neste cenário, não havia a preocupação com a segurança que hoje se faz necessária. Ao longo dos anos, com a difusão do uso, a integração de redes publicamente acessíveis, e o uso comercial destas redes como um sistema de comunicação pratico, rápido e confiável, a segurança dos dados transitados passou a ser de máxima importância [4]. A segurança de redes é se tornou um requisito essencial para sua utilização. Cada vez mais, os usuários demandam privacidade, e isso implica que seus dados precisem ser transferidos e armazenados de forma segura. As ações mais cotidianas se valem de recursos de segurança para manter senhas de acesso e arquivos pessoais sendo trafegados longe de usuários maliciosos. Essa preocupação se torna ainda maior quando olhamos para o mundo corporativo, onde projetos confidenciais que são potenciais fontes de lucro para as organizações devem ser protegidos de concorrentes dispostos a utilizar de recursos de ética questionável para conseguir resultados mais expressivos.

Um ataque que tem crescido recentemente é o *Ransomware* [5], onde o computador infectado tem seus arquivos criptografados e para obter acesso novamente o usuário deve pagar uma quantia ao autor do ataque. Existem diversos tipos de ataques que podem ser utilizados em uma rede de computadores, alguns exemplos são: *Malware [6], Spyware [7], Adware [8], Trojan [9], Worms [10] e Bots [11].*

A forma de infectar os computadores também é variada, todo tipo de troca de informação se torna um possível vetor de ataque. Pen Drives, CD-ROM, arquivos enviados por e-mail, arquivos baixados da internet, ou ataque a falhas em programas que os deixam expostos diretamente a Internet são alguns dos exemplos. O foco deste trabalho é o

último destes, pois é onde um controle rigoroso dos tipos de acesso e do comportamento mais se aplicam.

Existem diversas ferramentas para ajudar a proteção dos computadores por este meio. É comum ter-se um sistema antivírus instalado nos computadores. Os antivírus fazem o trabalho de proteção para os computadores, mas não são suficientes para cobrir completamente os vetores de ataque já conhecidos. Como complemento às funcionalidades dos antivírus, também existem os *firewalls* , IDS [12] - *Intrusion Detection System*, IPS [13] - *Intrusion Prevention Systems* ou IDPS - *Intrusion Detection and Prevention systems*, NIDS [14] [15] - *Network Intrusion Detection Systems*, HIDS [16] - *Host Intrusion Detection Systems*.

No contexto que estamos apresentando, a utilização de IDPS e *Firewalls* terá maior relevância, pois estas ferramentas tratam da coleta, analise e bloqueio do tráfego na rede de computadores.

2.1 *Firewall*

O *firewall* é uma forma de aumentar a segurança da rede de computadores ou de um computador. Pode ser implementado via *hardware* ou *software*. Seu funcionamento é verificar o tráfego *Transmission Control Protocol* - TCP e *User Datagram Protocol* - UDP da rede. Sua tradução literal significa "Parede de Fogo", pois seu objetivo é fazer o bloqueio e/ou liberação dos pacotes de rede baseados em regras estáticas previamente configuradas.

Com isso, o *firewall* consegue fazer uma proteção simples da rede e dos computadores. Seu uso mais comum é fazer o bloqueio de todas as tentativas de conexões ao computador pessoal, de certa forma, isolando o computador de ataques externos. É importante dizer que geralmente conexões provenientes do computador local para a rede externa são permitidas, assim, um computador infectado com um programa malicioso, pode enviar tráfego ou fazer ataques a outros computadores em sua rede.

É possível fazer diversos tipos de combinações de bloqueio e permissões, no entanto, apenas regras estáticas não são tão eficazes em grandes redes de computadores.

Para auxiliar os *firewalls* vieram os IDS.

2.2 *Intrusion Detection System*

Os IDSs, em sua tradução literal "Sistema de detecção de intrusão", têm como principal objetivo detectar acessos não autorizados à rede. Podem ser implementados via *hardware* ou *software*.

Os IDS são divididos basicamente em duas categorias: os HIDS e os NIDS. O objetivo de cada um deles, naturalmente, é detectar intrusos. No entanto, cada um deles tem um ambiente diferente para ser analisado. O NIDS verifica a rede como um todo, e geralmente recebe o tráfego de rede em pontos de agregação da rede, o HIDS se preocupa com apenas uma máquina, comumente são instalados agentes nas máquinas que serão analisadas e as informações são enviadas para um servidor que irá tratar esses dados. Isso é feito pois a análise das informações necessita de bastante capacidade computacional, e poderia atrapalhar o desempenho da máquina. A Figura 2.1 exemplifica ambas arquiteturas.

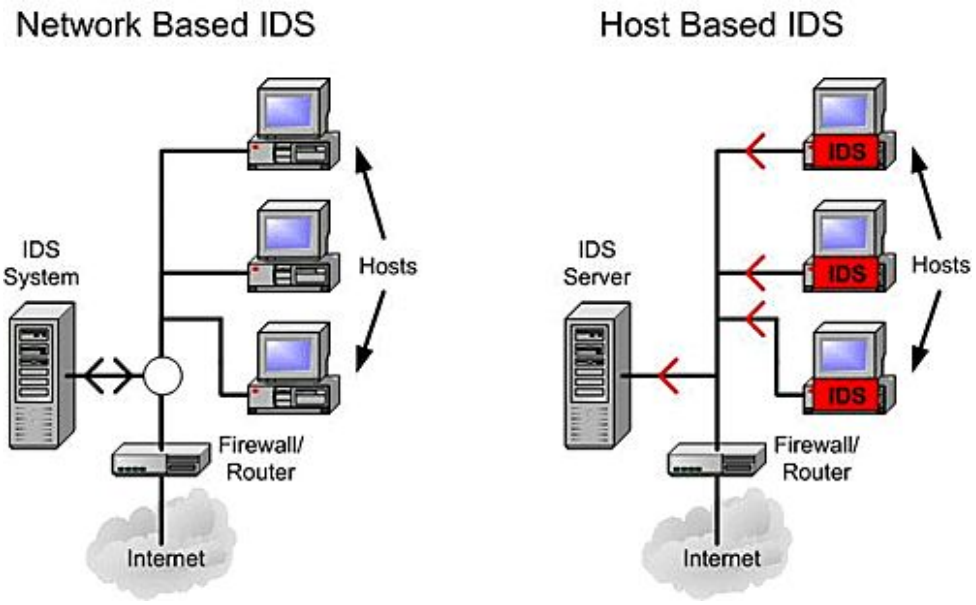

Figura 2.1: Comparação das arquiteturas de NIDS e HIDS [17].

Antes dos IDSs, os operadores e gerentes de redes, para detectar um intruso, precisavam notar comportamentos não usuais da rede ou de reclamações de usuários. Dado esse fator, muitos problemas às vezes demoravam a ser percebidos ou nem se quer eram detectados. Os intrusos podiam ficar por um tempo indeterminado nas redes de computadores ou nas máquinas de usuários.

Os IDSs, após a detecção de fatores que possam indicar um intruso, alertam os gerentes de redes de computadores ou de sistemas operacionais. Geralmente, esses alertas contêm informações necessárias para serem analisados de forma mais profunda pelos ge-

rentes.

Assim a probabilidade da rede estar mais segura é grande, pois agora gerentes e operadores analisam os problemas de forma mais ativa, uma vez que são notificados de possíveis problemas.

Os IDSs são mais comumente implementados baseados em assinaturas [18] e estatísticas/heurística de respostas.

As assinaturas funcionam como uma lista de assinaturas de ataques conhecidos. A ideia é semelhante a do funcionamento dos Antivírus. Algumas vezes fica difícil ver as diferenças entre as ferramentas, mas no geral, o IDS é mais abrangente que o antivírus, embora, atualmente, os Antivírus têm feito todos os papéis desde o *firewall* até o IPS.

Essas assinaturas, conforme os ataques são conhecidos, vão sendo geradas. Os IDSs analisam o tráfego da rede buscando por essas assinaturas. A maior crítica para o uso das assinaturas é que só funcionam para ataques conhecidos. Novos ataques não podem ser detectados nessa forma de implementação.

As assinaturas são um conjunto de texto ou código, que caso exista nos pacotes, indica que uma ação foi tomada. Como exemplo, uma assinatura referente a um ataque conhecido, como o c99shell.php [19], seria analisar os pacotes referente ao protocolo HTTP [20], mais especificamente o arquivo solicitado pela requisição HTTP e procurar pelo texto c99shell, em caso positivo, é provável que exista uma requisição ao c99shell. É importante que isso apenas indica a tentativa de requisição, não quer dizer que o arquivo exista ou o servidor esteja infectado, porém é um ótimo indicativo que algo suspeito está acontecendo.

A técnica por estatística e heurística se baseia no passado e em condições ensinadas ao programa. Dessa forma, o programa tem conhecimento de como o sistema deveria se comportar em certas ocasiões. Assim, quando o sistema detectar algo diferente do que considera normal, ele gerará um alerta para que o evento seja analisado.

Um dos problemas dessa abordagem de estatística é que, todos casos não ensinados, vão ser tratados como alertas. A utilização de estatística e heurística, geralmente está ligado a período de treinamento do programa. Um exemplo de problemas é: um servidor HTTP tem, em geral, seu tráfego conhecido durante um período, mas é possível que exista algum evento ou notícia que façam que o servidor seja muito mais acessado, gerando um alerta falso de ataque.

Algumas diferenças entre os HIDS e os NIDS por exemplo, é que os NIDS analisam e guardam as estatísticas de pacotes de redes de várias máquinas, também não verificam o sistema de arquivos a procura de intrusos, já os HIDS analisam os sistema de arquivos, programas em execução dentre outros.

Uma informação importante é que os IDS não executam nenhuma ação para tentar prevenir o ataque, apenas notificam possíveis problemas.

Antigamente, esses sistemas geravam muitos falsos positivos (FP), que são alertas de um possível intruso onde, de fato, não existe intruso e também os falsos negativos (FN), que são casos de intrusão em que o sistema não alertou. Por outro lado, também existem os verdadeiros positivos (VP) , que é a detecção de um intruso quando, de fato, existe um intruso, e os verdadeiros negativos (VN), que é a classificação correta de eventos legítimos nos sistemas. A acurácia (Ac) desses sistemas é baseado nessas métricas.

A Formula 2.1 indica o número total de eventos classificados corretamente como instruções. [21]

$$Ac = \frac{VP + VN}{VP + FP + VN + FN} \tag{2.1}$$

A métrica de precisão ou taxa de detecção (TD), é expressa pela Formula 2.2

$$TD = \frac{VP}{VP + FN} \tag{2.2}$$

Conforme os programas foram sendo melhorados o número de falsos positivos começou a diminuir. Tendo o número menor de falsos positivos, os IDS começaram a evoluir para os IPS [22].

2.3 *Intrusion Prevention Systems*

Os IPSs têm como objetivo prevenir intrusos na rede. Pode-se dizer que ele é uma evolução dos IDS, pois os IPS também fazem a detecção de intrusos da mesma forma que os IDS, porém após a detecção, existe mais uma etapa, que é a de tentar prevenir a intrusão.

Existem diversas formas de prevenir a intrusão. Um método comumente utilizado é deixar o IPS em modo de linha de rede, ou seja, todos os pacotes de rede antes de

chegarem ao destino final, passam pelo IPS, e após a análise o pacote é encaminhado ou descartado, a Figura 2.2 exemplifica a arquitetura do IPS em modo linha de rede.

Figura 2.2: IPS em linha [21].

Outra abordagem é análise do tráfego por um sensor de rede, e executar uma ação externa de bloqueio, normalmente atuando no *firewall* da rede como pode ser visto na Figura 2.3. A abordagem desse trabalho será parecida com essa, porém em vez de bloquear no *firewall*, que está localizado em pontos de agregação de rede, os fluxos serão alterados para que o tráfego malicioso não passe pela rede pelo controlador OpenFlow no ponto mais próximo ao atacante, essa á uma vantagem das SDNs.

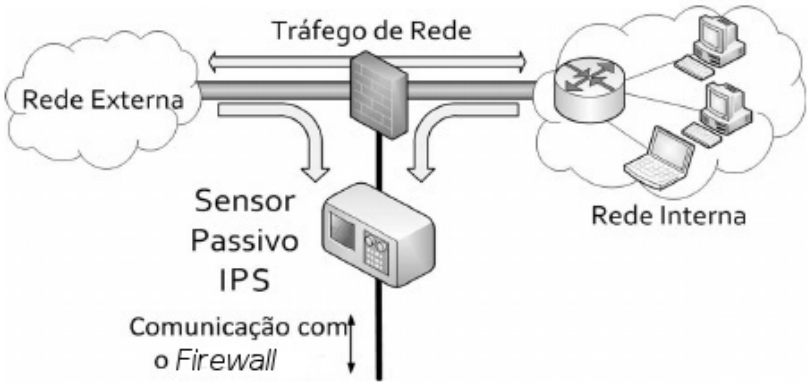

Figura 2.3: Sensor passivo de IDS [21].

É importante dizer que falsos positivos podem acarretar muitos problemas para a rede. Um trafego não malicioso sendo bloqueado pode ser bastante problemático para a rede.

Existem diversas frentes de pesquisas para aumentar a taxa de detecção e bloqueio e diminuir as taxas de falsos-positivos e falsos-negativos. Propostas recentes utilizam aprendizado de máquina [23], redes neurais [24], inteligência artificial [25], entre outras.

Apesar de alguns considerarem o IPS uma evolução dos IDS, às vezes é necessário manter ambas as ferramentas em sua rede. A utilização dos IDS pode ser apenas para

fazer uma análise da rede de computadores como um todo. Muitas vezes, o trafego da rede é redirecionado para o IDS em pontos estratégicos onde há maior tráfego. Já a utilização do IPS, se a acurácia não for alta, pode comprometer a entrega legítima de pacote na rede.

2.4 Programas

Os IDPS de código abertos mais utilizados são o Snort [26] [27], o Suricata [28], e o Bro IDS [29]. Os dois primeiros, são ferramentas utilizadas em redes de produção, enquanto o Bro IDS é mais utilizado em pesquisas. Nesta seção serão analisados alguns dos Sistemas de Detecção de Intrusão de código abertos.

2.4.1 Suricata

Oficialmente lançado em julho de 2010 é um programa de código aberto e é atualmente mantido pela *Open Information Security Foundation* - OISF. Alguns de seus benefícios são: processamento *multithread*, utilização do CUDA [30, 31] da NVIDIA [32] para computação paralela em processamento Gráfico ou *Graphic Processor Unit* - GPU, também utiliza aceleração via *hardware*.

O Suricata realiza detecção de intrusão baseadas em anomalia e em assinaturas. As assinaturas desenvolvidas para o Snort podem ser utilizadas diretamente ou otimizadas para o uso do motor de detecção do Suricata.

2.4.2 Bro IDS

O Bro IDS é descrito em [29] como um sistema monitor de rede independente para inspecionar o tráfego de um enlace de rede em tempo real. Seu desenvolvimento foi iniciado por Vern Paxson em Berkeley, CA.

È possível utilizar o Bro como um NIDS, inspecionando pacote por pacote que é trafegado pela rede de computadores. Um dos diferencias do Bro é que as politicas são escritas em linguagem própria, as regras de detecção são muito efetivas, porém, a complexidade de escrita é maior do que a Snort e Suricata. Devido a linguagem própria de detecção, é possível utiliza-lo para detectar assinaturas ou anomalias. Foi projetado para utilização de um único processo, no entanto existem implementações *multithread*.

2.4.3 Snort

O Snort é uma ferramenta de Detecção e Prevenção de Intrusão baseada em rede. Ele realiza a inspeção e monitoramento a nível de pacotes. Assim como o Suricata, e as detecções de ataques podem ser por assinatura de pacote ou anomalias. Uma das principais vantagens do Snort é a capacidade de fazer busca de sequência de caracteres e um código simples para especificar e escrever as sequências.

Devido a sua grande comunidade, existem muitas regras e assinaturas para detecção de ataques. O Snort pode ser executado em modo em linha ou utilizando alertas.

Esse trabalho utiliza o Snort como IDS, devido a seu grande número de regras e assinaturas, estabilidade e a implementação inicial de uma comunicação com o controlador OpenFlow escolhido.

Capítulo 3

Redes Definidas por Software

As redes de computadores, desde sua concepção inicial [33], são baseadas no uso de equipamentos capazes de tomar decisões de forma descentralizada. Neste contexto, cada hardware que é adicionado contém também um software capaz de implementar toda a lógica necessária para fazer sua integração ao meio. No entanto, esse tipo de implementação traz consigo algumas dificuldades, como a falta de padronização de algumas implementações e o uso de protocolos proprietários que não podem ser substituídos por novos softwares escolhidos pelo gerente da rede, o que causa problemas de compatibilidade entre fabricantes e, principalmente, acarreta uma maior dificuldade de manutenção e gerência em redes com grandes quantidades de equipamentos.

Atualmente, a demanda por um ambiente que acompanhe a velocidade dos acontecimentos, através de uma maior facilidade e agilidade para monitorar e aplicar novas configurações nos elementos da rede tem sido uma tendência. Desta necessidade, começaram a surgir propostas para que os planos de dados e de controle fiquem fisicamente separados, mudando a concepção inicial das redes onde os elementos funcionavam de forma independente e distribuída. Esse é o princípio básico das SDNs - *Software-Defined Networking*

Uma forma de controle da rede para a implementação das SDNs é o uso de um plano de controle centralizado, apesar do plano de dados continuar fisicamente distribuído. Uma das primeiras propostas nesse sentido foi o SANE [34] - *Secure Architecture for the Networked Enterprise* e o Ethane [35]. A partir deles, o conceito de Redes definidas por Software foi sendo realmente definido e saindo do plano teórico para o mundo prático.

As SDNs [36] tem ganhado muita atenção da comunidade acadêmica e de indústrias na área da computação e redes. Grandes empresas como Google, Facebook, Microsoft e

Yahoo, criaram a *Open Networking Foundation* (ONF) [37], com o principal objetivo de incentivar a adoção da SDN como padrão aberto de definição e para facilitar o desenvolvimento desta vertente da área de redes de computadores.

3.1 Conceito

Pela definição de Nadeau, "Redes Definidas Por Software são uma abordagem de arquitetura que otimiza e simplifica as operações de rede por proporcionar a interação mais próxima entre as aplicações, serviços e equipamentos, sejam reais ou virtualizados." [38]

A maior distância conceitual entre as redes tradicionais de computadores, que atualmente são amplamente usadas no mercado, e as SDNs é a diferenciação que a segunda possui entre o plano de gerência e plano de dados. Na abordagem atual, tem-se apenas um sistema que é capaz de, independentemente, realizar as funções para as quais foi configurado e trabalha com ciência e informações dos elementos diretamente conectados a ele.

Nas implementações mais comuns das SDNs, todos os elementos estão em contato com o sistema de controle, e este, por sua vez, tem ciência de toda a rede e das informações a ele reportadas. Com base nestas informações, pode tomar ações e encaminhar fluxos pelos caminhos mais indicados. [39]

3.1.1 Plano de controle

O plano de controle é o domínio responsável pela lógica a ser implementada. Por ele, todos os elementos da rede irão enviar fluxos desconhecidos até o controlador da rede, para assim, o controlador identificar e tomar decisões sobre como proceder de acordo com suas regras e configurar os fluxos. Como toda a parte lógica gera um questionamento ao controlador, a configuração dos fluxos e suas rotas ótimas fica a cargo dele, gerando a facilidade de um ponto único de configuração. Uma grande vantagem, é ter a noção do todo no momento de tomar as decisões, assim possibilitando variar o comportamento de acordo com o estado atual da rede.

É importante ressaltar que o controlador não é centralizado ou único, ele é uma parte do plano de controle. As funções do plano de controle em uma analogia com a rede atual de dados são:

- estabilização do conjunto de dados locais;
- manter o conjunto de dados;
- manutenção da *Routing Information Base* – RIB; e
- manutenção da *Forwarding Information Base* – FIB.

3.1.2 Plano de dados

O plano de dados é responsável apenas pelo encaminhamento dos pacotes. É importante reforçar que o equipamento responsável pelo encaminhamento não mais participa da decisão de qual regra será criada, ele apenas possui uma tabela, FIB, a qual, quando confrontada com as características do pacote recebido, descreverá o comportamento que a ele deve ser aplicado.

As decisões de encaminhamento são regras simples, baseadas em parâmetros dos pacotes. Caso o equipamento não tenha regra definida para o pacote, esse pacote, em geral, é enviado para o controlador, onde o plano de controle entra em ação. Os pacotes sem regra, também podem ser descartados, dependendo da implementação da SDN.

As funções do plano de dados incluem:

- encaminhar pacotes baseados na FIB; e
- encaminhar pacotes para o plano de controle.

3.1.3 Vantagens das SDNs

As principais vantagens trazidas por uma SDN podem ser simplificadas [36] como:

- Programação direta: o controle da rede pode ser diretamente programado, pois é separado do encaminhamento de pacotes.
- Ágil: abstraindo o controle do encaminhamento de pacotes, possibilita os administradores ajustarem o tráfego da rede de computadores conforme necessário.
- Possibilidade de controle centralizado: a inteligência da rede pode ser logicamente centralizada em um programa baseado em controladores de SDN que mantém uma visão global da rede de computadores, dessa forma as regras das aplicações visualizam a rede como um grande e único *switch* lógico.

- Configurável por programação: SDN possibilita os administradores de rede otimizarem e gerenciarem a rede muito rapidamente com regras dinâmicas e automáticas.
- Agnóstica a fabricantes: Se implementada utilizando um padrão aberto, as SDNs possibilitam a integração de vários fabricantes, uma vez que não dependem de programas proprietários.

A Figura 3.1 demonstra a arquitetura básica de uma rede SDN, assim é possível ter uma visão melhor de como os componentes estão interligados.

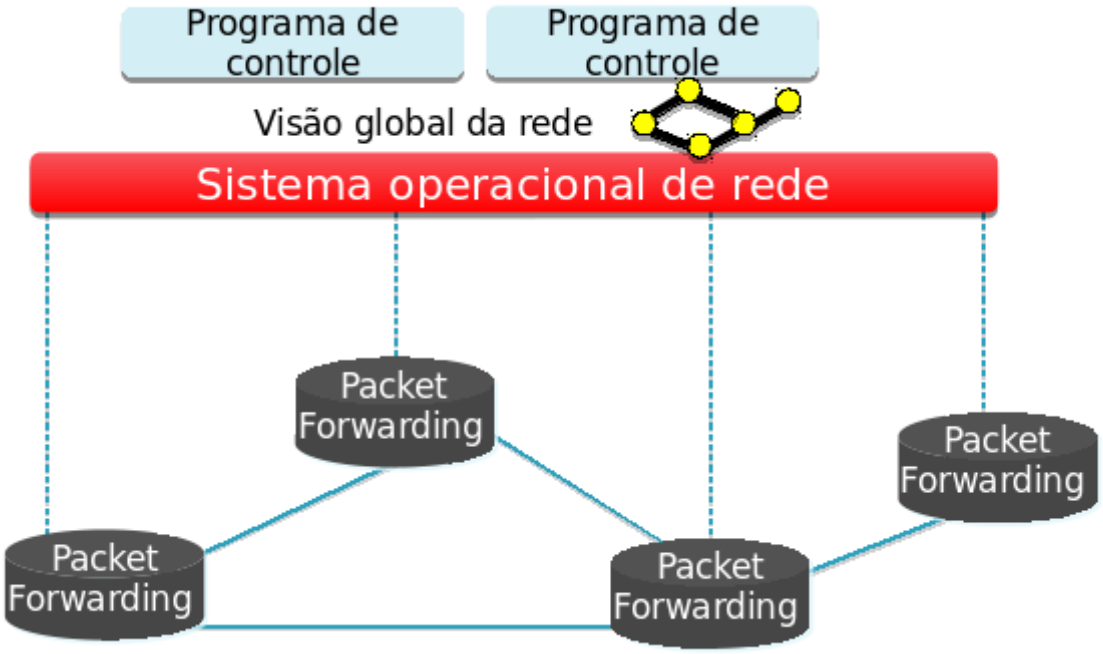

Figura 3.1: Arquitetura das SDN, nas quais os planos de dados e de controle são fisicamente separados. [40]

Também pode-se fazer uma comparação simples entre a diferença de arquitetura das redes atuais com relação às redes definidas por software para entender a causa pela qual as SDNs simplificam a inovação. A inovação é mais simples e rápida nas redes definidas por software por terem uma interface aberta, como mostra a Figura 3.2.

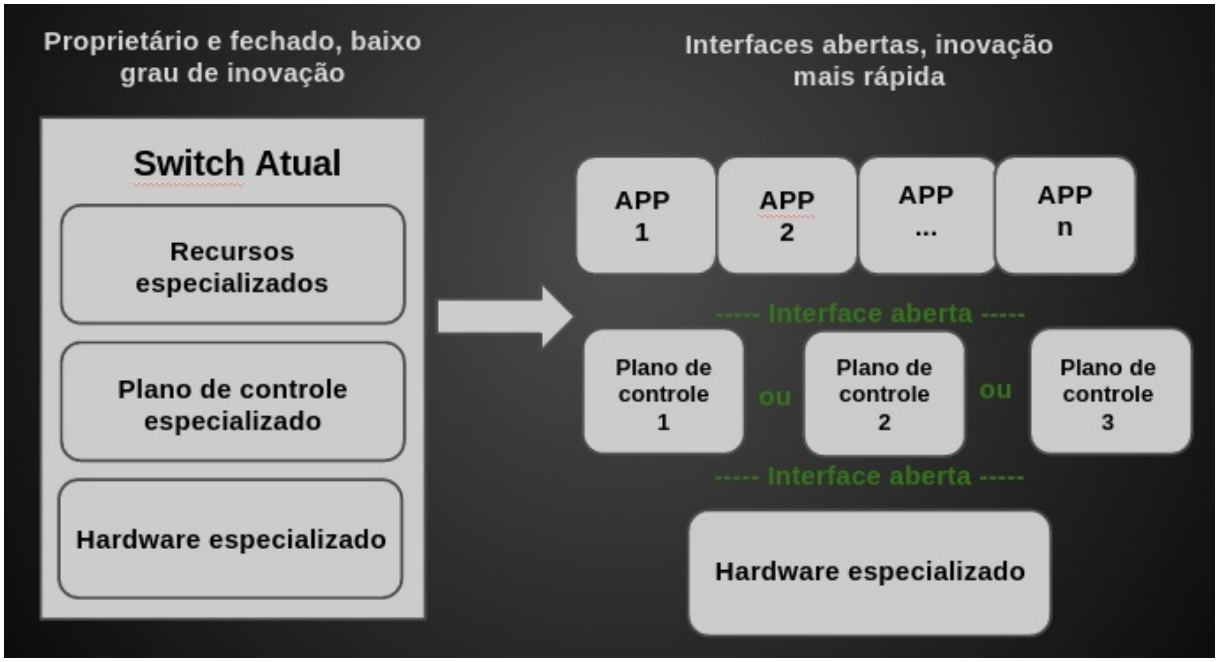

Figura 3.2: Comparação da arquitetura da rede atual e da arquitetura da SDN. [40]

3.2 OpenFlow

As SDNs são um conceito, mas é preciso ter um padrão para sua implementação. O OpenFlow [41, 3, 42] foi o primeiro padrão de interface de comunicação definido entre o plano de controle e o plano de dados de uma rede SDN. Ele define a forma de acesso direto e de manipulação do plano de dados dos equipamentos de rede, como roteadores e *switches*.

Até a presente data, diversas versões do OpenFlow já haviam sido publicadas. A primeira versão do OpenFlow foi a 0.8 [43] lançada em 5 de Maio de 2008, em 20 de Julho de 2009 a versão 0.9 [43] do protocolo foi lançada. A primeira versão estável do OpenFlow, a 1.0, foi lancada no dia 31 de Dezembro de 2009 [44], após o lançamento da primeira versão estável, outras versões vieram a ser lançadas. A versão 1.1 do protocolo foi lançada em 28 de Fevereiro de 2011 [45]. Em 05 de Dezembro 2011, a versão 1.2 [46] foi disponibilizada, em 25 Junho de 2012 a versão 1.3 [47], em 14 de Outubro 2013 veio a versão 1.4 [48], até a última versão disponível 1.5 [49] que foi lançada em 09 Janeiro 2015.

A cada lançamento, novos meios de controle e funções são adicionados ao protocolo, permitindo assim outras abordagens para o tratamento dos fluxos. Esse desenvolvimento contínuo do pilar da SDN, que está aquecido, é um grande diferencial. Além de melhorias, o que já foi lançado também é alvo de manutenção. Versões corretivas são lançadas sempre

que necessário.

3.2.1 Conceitos

O OpenFlow tem três componentes básicos, o *swicth* OpenFlow, o controlador OpenFlow e o canal seguro de comunicação OpenFlow. Estes componentes em conjunto exercem as funções necessárias para o funcionamento básico das redes definidas por software. A Figura 3.3 exemplifica os componentes.

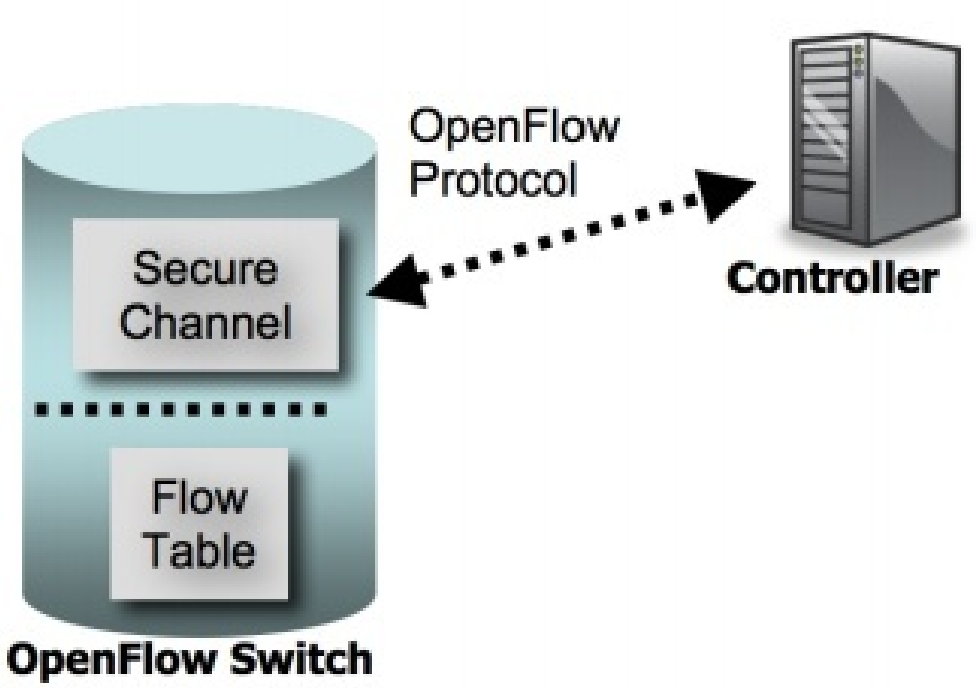

Figura 3.3: Componente básicos do OpenFlow [44]

O plano de dados é implementado pelo *switch* OpenFlow. Como o plano de dados não tem inteligência para decidir o que fazer com os pacotes recebidos, ele depende do controlador para a tomada de decisão. Pode-se dizer que o *switch* OpenFlow é uma abstração para os *switches*, roteadores e pontos de acesso Wifi, pois em uma rede SDN, esses componentes seriam um *switch* OpenFlow, e a lógica da rede estaria implementada no controlador OpenFlow. A comunicação entre estes elementos é realizada através do canal seguro de comunicação OpenFlow. Assim, o controlador OpenFlow e o *switch* OpenFlow podem trocar configurações e dados, sempre utilizando o protocolo OpenFlow durante a comunicação.

3.2.1.1 Canal seguro de comunicação OpenFlow

O canal seguro de comunicação é o meio utilizado para a comunicação entre *switch* OpenFlow e o controlador OpenFlow. Ele permite a troca de comandos e pacotes entre esses dois elementos. Por se tratar de parte crucial em um sistema distribuído, a comunicação entre os elementos deve ser altamente segura, evitando que este seja um vetor de ataques de elementos mal-intencionados na rede. Para dar confiabilidade ao canal, a interface de acesso recomendada é o protocolo *Secure Socket Layer* - SSL, já amplamente utilizado, que utiliza a encriptação dos dados trafegados utilizando certificados confiáveis reconhecidos pelos participantes da transação e que, em teoria, não podem ser quebrados.

A comunicação entre o controlador e o *switch* é feita integralmente utilizando o Protocolo OpenFlow. Assim como em outros protocolos, nele existem alguns tipos básicos de mensagens que podem ser trocadas entre os elementos. Essas mensagens são classificadas como:

- Simétricas: São mensagens geradas sem solicitação dos elementos, em qulaquer direção. Exemplos são as mensagens *hello* e *echo*. A primeira é trocada entre o controlador e o *switch* na inicialização da rede, a segunda é usada, principalmente, para verificar se a conexão entre o *switch* e o controlador continua ativa e para identificação de latência e banda.

- Assíncronas: São mensagens enviadas pelo *switch* sem a solicitação do controlador. Informam eventos na rede, erros, mudanças no estado do *switch* e chegada de pacotes. Um exemplo desse tipo de mensagem é a *Packet In*, que notifica a chegada de um fluxo não configurado no *switch*.

- Controlador para *switch*: São mensagens iniciadas pelo controlador, usadas para gerenciar diretamente ou inspecionar o estado do *switch*. Estas mensagens permitem que o controlador configure o *switch*, modifique estados e entradas de fluxo, dentre outras características.

3.2.1.2 *Switch* OpenFlow

O *switch* OpenFlow é, de certa forma, apenas uma parte dos equipamentos de comutação de pacotes que temos atualmente. Porém, dentre suas peculiaridades, vale frisar o conceito de fluxo. O fluxo é um conjunto de características de determinados pacotes na rede. Por exemplo, um fluxo pode ser todo tráfego HTTP, ou todo o tráfego proveniente

de um endereço IP. Estes parâmetros são definidos no momento da criação da regra e, a cada recebimento de pacote, este será comparado as regras existentes.

Estes elementos possuem uma tabela de fluxos. Essa tabela define como os pacotes que chegam ao *switch* devem ser tratados, de forma muito similar as tabelas de roteamento hoje existentes. Ela também é uma lista, porém o número de parâmetros analisados para a caracterização de um pacote como pertencente a um fluxo não é mais apenas o seu destino. A tabela de encaminhamento do OpenFlow possui múltiplos parâmetros, os quais entramos em maiores detalhes na Figura 3.4. Além disso, a forma como o pacote é processado também varia. Nos roteadores, utiliza-se o *best match*, enquanto no OpenFlow busca-se a regra de maior prioridade que define os campos do pacote a ser encaminhado.

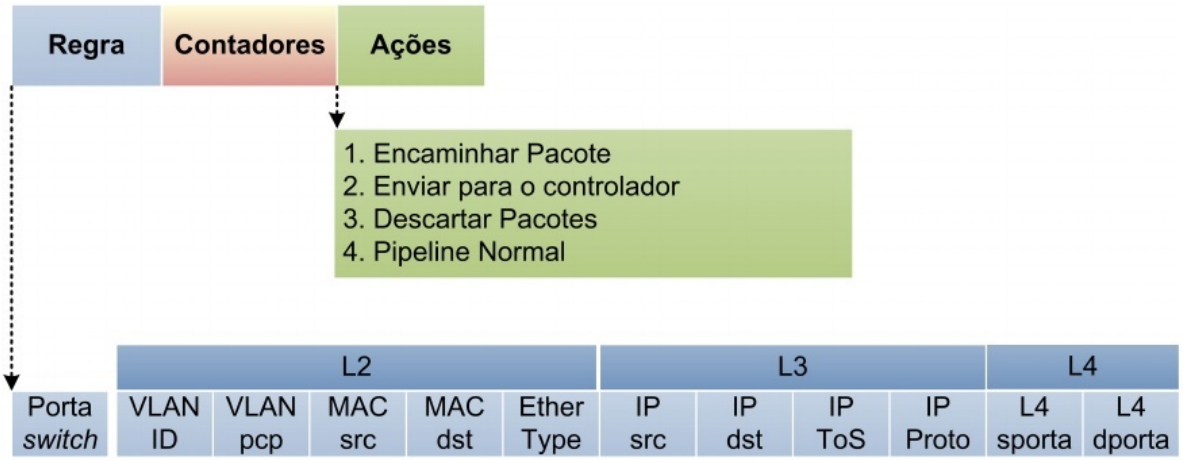

Figura 3.4: Formato da tabela de fluxos do OpenFlow versão 1.0 [44].

Na versão 1.0 do OpenFlow, existem três campos na tabela de fluxos, as regras, os contadores e as ações, como mostrado na Figura 3.5. A tabela é única nessa versão do protocolo. A partir da versão 1.3 do OpenFlow, é possível ter múltiplas tabelas de fluxos, isso adiciona grande flexibilidade ao OpenFlow [50], pois agora é possível adicionar ações ao pacote em uma tabela, e em seguida, se necessário, encaminha-ló para outra tabela de fluxo, onde novas ações podem ser adicionadas, assim é possível adicionar diversas ações com maior flexibilidade, ao final do processo, todas ações são executadas.

A regra é formada a partir de um ou mais campos do cabeçalho do pacote. No OpenFlow 1.0, existem 12 campos que podem ser usados no cabeçalho, como mostrado na Figura 3.4. Assim, um fluxo pode ser identificado por uma combinação desses campos. Vale ressaltar que o *switch* utiliza campos de diferentes camadas de rede, o que aumenta sua flexibilidade para encaminhamento e detecção de pacotes.

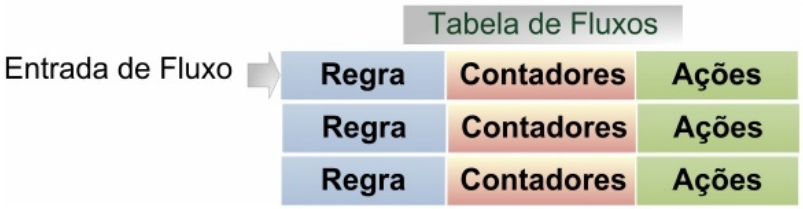

Figura 3.5: Tabela de Fluxos [44].

Os contadores são estatísticas referentes aos fluxos do *switch*. Alguns exemplos são: número de pacotes e *bytes* que passaram por cada fluxo, o tempo desde que o fluxo foi configurado no *switch* e o tempo decorrido desde a última vez que um pacote desse fluxo foi identificado pelo *switch*. Um detalhe importante é que os contadores são reiniciados automaticamente ao atingirem o valor máximo, sem qualquer aviso. Por isso, é necessário que o controlador seja responsável por esse controle.

Conforme as versões do OpenFlow vão sendo lançadas, maiores são a flexibilidade de alterações e configurações de fluxos. Por exemplo, na versão 1.1 do OpenFlow, existem mais campos no cabeçalho. Dentre eles, é possível a verificação do *Multiprotocol Label Switching* – MPLS no encaminhamento dos pacotes.

Por se tratar de um protocolo em constante desenvolvimento, novas funcionalidades que são importantes para aumentar sua utilização foram adicionadas na versão 1.3 do OpenFlow. Nela, também foi adicionado o suporte a *Per flow meters*, que dá a possibilidade de limitar as taxas de pacotes enviados por um fluxo. Com isso, é possível limitar o envio de muitos pacotes para o controlador e para o IDS, evitando a sobrecarga, assim como limitar a utilização de máquinas suspeitas.

3.2.1.3 Controlador OpenFlow

Ao remover o poder de decisão dos elementos que compõem as redes, este deve ser exercido por outro componente. No caso do OpenFlow, o responsável pelas decisões é o controlador OpenFlow. Trata-se de um software, que implementa o protocolo OpenFlow, tornando viável o gerenciamento centralizado da rede. Ele é o responsável por monitorar, gerenciar e tomar decisões referente à rede de dados. Pacotes não reconhecidos na tabela de fluxos dos *switches* OpenFlow são enviados para o controlador, para que este possa tratar e tomar alguma ação levando em consideração a lógica pré programada.

Pode-se dizer que o controlador OpenFlow atua como um Sistema Operacional de

rede [51], fazendo assim, uma analogia os Sistemas Operacionais em computadores. Sistemas operacionais oferecem uma interface de alto nível para a utilização de recursos do *hardware* pelas aplicações. Essa é a principal função do controlador OpenFlow.

3.2.2 Funcionamento

Tendo em mente os conceitos estruturais de cada elemento, exprime-se também a forma de atuação de cada componente de uma rede OpenFlow, mostrando o passo a passo durante o processamento e encaminhamento de fluxos.

Após a montagem física e o estabelecimento dos canais seguros de comunicação entre *switch* e controlador, a cada pacote que chega ao *switch* OpenFlow, ele faz a separação dos cabeçalhos e verifica se existe alguma entrada correspondente em suas tabelas de fluxos. Caso exista, o *switch* aplica a ação. Caso não exista, ele envia o pacote para o controlador pelo canal seguro e aguarda instruções. [1]

O controlador recebe o pacote, faz o tratamento que foi previamente estabelecido em sua configuração para este padrão e baseado em características do pacote, aplica uma ou mais regras nos *switches*. O comportamento do *switch* durante este processamento do controlador é configurável. Com a premissa de ser um tratamento rápido, é comum que o switch espere a resposta, mantendo os pacotes originais em *buffer* para que sejam encaminhados após a geração da regra.

A Figura 3.6 demonstra o funcionamento básico de um *switch* OpenFlow 1.0.

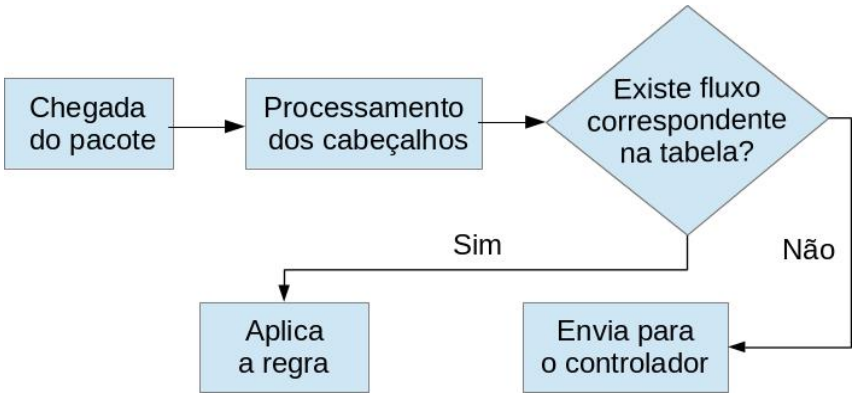

Figura 3.6: Processamento dos Fluxos nos *swicthes*.

O funcionamento básico de um controlador é exemplificado na Figura 3.7. Na parte

[1] O caso do *switch* enviar o pacote para o controlador é padrão e obrigatório na versão 1.0 do OpenFlow, porém com as novas versões, é possível mudar esse comportamento, como por exemplo, descartar o pacote ou enviar ele para uma tabela de fluxos diferente.

da decisão da ação a ser tomada, ficam os algoritmos desenvolvidos para controlar aquela rede específica, dando flexibilidade para o controle da rede. É importante ressaltar que controladores, em geral, são executados em PCs, de forma que é mais fácil adicionar códigos novos aos controladores do que aos roteadores atuais.

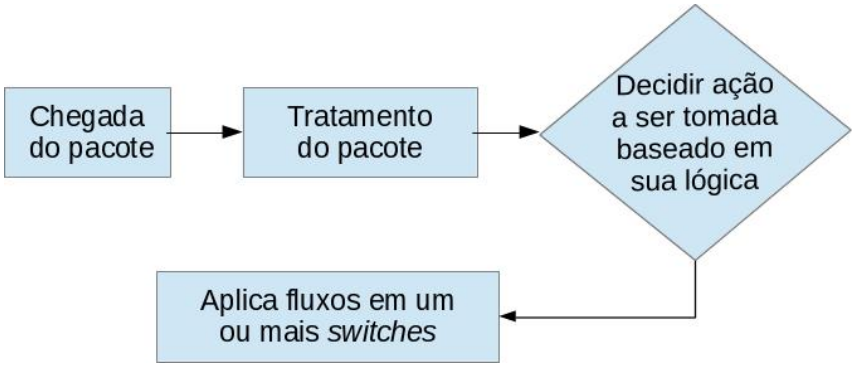

Figura 3.7: Processamento de pacotes pelo controlador OpenFlow

3.2.3 Implementações de controladores OpenFlow

Por se tratar de um software, existem diversas implementações de controladores OpenFlow disponíveis, em linguagens e com focos diferentes. A arquitetura da plataforma foi desenvolvida com o objetivo de incentivar a inovação. Padrões de comportamento e interação regem as interconexões, mas todas as funcionalidades extras estão abertas a serem exploradas.

Abaixo, serão descritos brevemente os controladores OpenFlow que tem sido mais utilizados em trabalhos, pesquisas e pequenas implementações de redes deste tipo.

3.2.3.1 OVS-Controller

O OVS-controller é controlador de referência OpenFlow. Ele é bem simples, propositalmente. O seu objetivo principal é fornecer uma referência para outras implementações na forma de utilização das funcionalidades e dos requisitos necessários na implementação de um controlador OpenFlow.

Por ser apenas uma referência, as funcionalidades necessárias para a execução dos testes pensados para validação da proposta ficariam a quem do necessário. Por isso, não era possível utilizar esse controlador para implementar a proposta desse trabalho.

3.2.3.2 NOX

O NOX [52] foi um dos primeiros controladores OpenFlow. Ele introduziu a ideia do sistema operacional de rede. Escrito em C++, é focado em velocidade de processamento dos fluxos.

Embora muito rápido e eficiente, não foi o controlador utilizado para esse trabalho. Seu desenvolvimento aparenta ter parado no final de 2013 [53]. Além disso, a documentação disponível não tratava de funções que seriam necessárias para atender as condições impostas pelos cenários de teste.

3.2.3.3 POX

O POX [54] é um controlador que surgiu de diversas APIs (*Application Programming Interface*) utilizadas no NOX. Inicialmente, ele era recomendado para experimentos educacionais, pois sua interface era bem mais amigável e elegante do que a do NOX. Foi verificada a possibilidade da utilização do POX para essa dissertação. Todavia, o suporte ao protocolo OpenFlow parou no 1.0, fato este que inviabiliza a utilização do mesmo para o trabalho proposto. Isso se deve também ao fato do seu desenvolvimento ter se encerrado no final de 2013 [55].

3.2.3.4 Floodlight

O controlador OpenFlow Floodlight [56], é escrito em JAVA [57], tem suporte a OpenFlow 1.0 e 1.3 e continua com seu desenvolvimento ativo de acordo com seu repositório de código [58].

Verificando a possibilidade de sua utilização, ele foi descartado por possuir uma curva de aprendizado grande. Sendo assim, o Floodlight também não atendia aos requisitos para a execução dos experimentos propostos. Vale notar que ele pode ser uma opção viável para outros trabalhos relacionados.

3.2.3.5 Ryu

O Ryu [59] é um controlador de SDN, implementado na linguagem de programação python e é um programa de código aberto. O Ryu é utilizado para gerenciamento de redes e aplicações de controle, um de seus pontos fortes é o suporte a vários protocolos tais como

o OpenFlow , *Network Configuration Protocol* (NETCONF), *Configuration Protocol* (OF -Config), e outros.

Seu desenvolvimento continua ativo, suporta as versões 1.0 a 1.5 do OpenFlow completamente e utiliza testes para verificar sua compatibilidade com OpenFlow *switches* além de testes unitários em seu código, dando maior confiabilidade em compatibilidade com as versões do OpenFlow e dos *switches* utilizados.

Devido a sua compatibilidade com os protocolos OpenFlow, seu desenvolvimento ativo, documentação atualizada e de fácil acesso, ele foi escolhido para ser utilizado nesse trabalho.

3.2.3.6 Outros

Além dos controladores aqui citados, existem muitos outros. Uma lista, mantida por Martin Casado, tenta manter as outras referências e nela é possível encontrar vários links para outros controladores OpenFlow [60].

3.2.4 OpenFlow *Switches*

Como dito anteriormente, em SDNs, além dos controladores, é necessário que existam os *switches* OpenFlow. Eles são responsáveis pela implementação do plano de dados. Os *switches* OpenFlow podem ser físicos, ou seja, um *hardware* específico para isso, ou serem executados via software em uma máquina virtual, emuladores ou simuladores. Para ser considerado como tal, é necessário que implemente o protocolo OpenFlow. Como já explicado anteriormente, existem diversas versões do protocolo OpenFlow, sendo assim, é muito importante verificar a compatibilidade dos *switches* com o controlador e com as funcionalidades que serão utilizadas na implementação já no momento da escolha dos elementos.

Por se tratar de uma prova de conceito, este trabalho utilizará apenas *switches* implementados via *software*, visando manter o investimento financeiro mínimo em hardware, além de uma maior facilidade nas alterações dos ambientes para os experimentos.

3.2.4.1 Open vSwitch

O Open vSwitch [61] é um *switch* via *software* escrito em C [62], que suporta os protocolos 1.0 até 1.4 do OpenFlow. Ele é estável para produção e seu desenvolvimento

continua ativo.

Por se tratar de um equipamento considerado estável para produção, foi utilizado durante o início dos experimentos, porém devido a algumas incompatibilidades com funcionalidades desejadas, que ainda não haviam sido implementadas no momento da execução dos testes, ele foi descartado. No caso, a funcionalidade que ele não suporta é a utilização dos *meters* do OpenFlow, que já está finalizada em sua definição, mas não estava implementado no Open vSwitch [63].

3.2.4.2 Switch desenvolvido pelo CPqD

O *switch* desenvolvido pelo Centro de Pesquisa e Desenvolvimento em Telecomunicações - CPqD [64] é baseado no Ericsson TrafficLab 1.1 [65] e foi modificado para ser compatível com a versão 1.3 do OpenFlow. Devido ao problema encontrado com o Open vSwitch, foi necessário utilizar um *switch* com mais funcionalidades. De acordo com a página de compatibilidade do Ryu [66], o CPqD se tratava de um *switch* com mais funcionalidades e teste funcionais que os outros.

Devido a sua facilidade de instalação e sua compatibilidade com os requisitos, ele foi escolhido para ser utilizado. Facilidade de acesso aos desenvolvedores também foi um fator que pesou na escolha deste *switch*.

Em Julho de 2016, foi enviada uma mensagem que seu desenvolvimento agora é baseado no melhor esforço, ou seja, não existe mais desenvolvimento ativo, embora ele continue bastante popular.

3.2.5 Ferramentas de emulação e simulação

Neste ambiente, com o controlador OpenFlow e o *switch* selecionado, seria possível criar uma rede física, ou se valer de uma ferramenta de emulação ou simulação de rede. Mantendo a linha da escolha dos *switches* virtuais, optou-se por um ambiente emulado para maior flexibilidade e agilidade durante a execução de experimentos envolvendo diferentes números de elementos.

Assim, como busca-se sempre uma diversidade de cenários, os emuladores e simuladores auxiliam nos experimentos de pesquisa. Um levantamento de ferramentas compatíveis com o SDN/OpenFlow foi feito e uma breve descrição sobre o que foi encontrado segue abaixo.

3.2.5.1 ns-3

O ns-3 [67] é um simulador de redes de código aberto. O objetivo desse simulador é ser um método flexível e popular entre pesquisadores em diferentes ambientes simulados ou emulados de rede.

Embora muito utilizado para simulações de rede, pouca documentação foi encontrada sobre a utilização de OpenFlow em conjunto ao ns-3, especialmente quando este fluxo é relacionado a controladores externos. Foram encontrados documentos e modelos para ns-3 utilizando um controlador interno simulado.

Devido a dificuldade de integração com um controlador externo, este simulador não se mostrou opção viável para esse trabalho.

3.2.5.2 EstiNet

EstiNet [68] é um simulador e emulador de redes SDN com a capacidade de utilização de controladores externos. Ele implementa as funcionalidades de forma abrangente, porém é um software proprietário e uma licença é necessária para a sua utilização.

Por ser um programa proprietário e necessitar a utilização de uma licença, ele não foi utilizado para esse trabalho.

O artigo [68], faz uma comparação entre o EstiNet e o Mininet, e conclui que o EstiNet pode trabalhar em modo simulador e emulador e é um emulador confiável mesmo utilizando grandes cargas de dados. Ainda na comparação do artigo, o Mininet funcionou bem no geral, no entanto, demostrou algumas inconsistências dependendo do cenário do teste.

3.2.5.3 Mininet

O Mininet [69] é um emulador de redes. Com ele, é possível a criação de redes de máquinas virtuais, *switches*, controladores e links. Isso torna possível o controle sobre o ambiente de testes de forma geral. Se trata de um programa de código aberto, existe grande documentação de sua utilização e integração com os *switches* OpenFlow e controladores externos e seu desenvolvimento continua ativo.

A criação de ambientes é muito rápida o que torna o tempo dos experimentos mais bem aproveitado, permitindo uma maior variedade de ambientes. Sua maior limitação é

que, atualmente, ele não pode exceder o uso de CPU e utilização de rede de um único servidor.

Devido à grande documentação do projeto sobre OpenFlow e sua boa integração com controladores e OpenFlow *switches*, ele foi escolhido para ser utilizado nesse trabalho.

Capítulo 4

Trabalhos Relacionados

Para uma ampla adoção das redes OpenFlow, é importante conhecer suas vulnerabilidades e desenvolver métodos para superá-las. Em [70], os autores fazem um breve levantamento sobre os problemas de segurança encontrados nas redes OpenFlow. Kloti et al. mostram uma análise de segurança e algumas vulnerabilidades graves das SDNs, dentre elas os ataques DoS - *Denial of Service* [71] e a manutenção do sigilo da informação transmitida. Os ataques DoS são mais preocupantes em um modelo onde existe um controlador centralizado, pois ele representa um possível gargalo na rede.

Embora o controlador seja um alvo central para ataques, eles também trazem vantagens em termos de segurança. Muitos autores estão tirando proveito da existência de um controlador e da visão centralizada da rede para propor novos métodos de análise e controle que não poderiam ser bem implementados de maneira distribuída [72], por exemplo, ataques DDoS - *Distributed Denial of Service*, são mais facilmente detectados com um controlador centralizado, pois muitos fluxos terão as mesmas características. O OpenFlow possibilita fazer uma alteração rápida na rede, baseada em análises da rede como um todo.

Há também autores trabalhando na adaptação e melhoria de sistemas de análise de tráfego e detecção de intrusão que já são bem conhecidos nas redes tradicionais [14].

Essas propostas também utilizam a capacidade de processamento do controlador e a facilidade em reconfigurar os dispositivos de encaminhamento de pacotes para tomar ações necessárias para prover a segurança da rede de forma automática [71]. Entre essas propostas, destacam-se o Snortflow [27] e o SDNIPS[73], que serão descritos em detalhes mais adiante.

4.1 Segurança em SDN

Existem trabalhos referentes a segurança de uma SDN. Esses trabalhos apontam que existem diversas formas de aumentar a segurança de uma rede, principalmente as SDN que dispõem de mecanismos mais dinâmicos. Alguns artigos, citados adiante, são focados em alterar o protocolo OpenFlow, outros em adicionar *middleboxes* na rede, dentre outros. O importante é saber que ainda existem muitos problemas de segurança para que a SDN seja confiável e robusta em produção.

O trabalho *A Comprehensive Security Architecture for SDN* [74] apresenta uma alteração no protocolo, onde é introduzido a ação "OFPAT_DETECTION". Nesse caso, os pacotes são enviados a uma porta especifica e podem ser tratados por alguma aplicação antes de chegarem ao controlador, essa abordagem é interessante, pois diminue a possibilidade de um ataque DoS aos controladores OpenFlow. Embora tenha o mesmo objetivo, a proposta dessa dissertação, ao invés de fazer uma alteração de protocolo, utiliza ações já existentes no OpenFlow. Dessa forma, a ideia apresentada pelo trabalho citado é implementada nesse trabalho de uma forma diferente, porém obtendo o mesmo resultado sem causar incompatibilidade com as versões já existentes de switches e controladores.

O artigo [2] levanta os problemas de segurança de uma SDN, mostrando que existem duas linhas de pensamento: a primeira mostra que os problemas de segurança podem ser resolvidos por programação, devido a visão centralizada da rede; a segunda linha diz que a mesma centralização e programação da rede trazem novos problemas de segurança. Em uma rede centralizada, fica mais simples fazer bloqueios pontuais em *switches* mais próximos a usuários, algo que não é possível em redes tradicionais, pois em geral, os *switches* não tem essa capacidade. Por outro lado, um atacante tem uma boa chance que seus ataques passem pelo controlador, abrindo um grande leque de opções. Alguns exemplos são injeção de código nos pacotes para análise ou execução no controlador, geração de pacotes mal formados para sobrecarregar o controlador, dentre outros.

4.2 IDS com SDN

Considerando o uso de IDS associado às SDNs, destaca-se o IPSFLOW [75], cuja proposta se assemelha bastante com esse trabalho, pois, utiliza um IPS juntamente com o controlador para analisar o tráfego da SDN. Contudo, o IPSFlow apenas trata da proposta sem nenhuma implementação. O IPSFLOW, assim como esse trabalho, possibilita a

utilização de vários IDS/IPS em conjunto com um controlador, analisando a diferença que a localização dos IDS pode fazer no sistema. O trabalho apresentado aqui pode ser considerado a continuação do IPSFLOW, pois aqui é demostrado o funcionamento da ideia apresentada no IPSFLOW com algumas novas funcionalidades, como a utilização do histórico de ataques de rede e portas de interesse, que são parâmetros utilizados nos algorítimos desenvolvidos nesse trabalho.

O SnortFlow[27] é uma proposta voltada para segurança de redes OpenFlow para infraestruturas virtualizadas, que permitem a construção de nuvens. A integração entre o Snort [26] e o controlador OpenFlow NOX é feita para permitir a aplicação das regras de segurança automaticamente na rede. No artigo, os autores mostram testes de desempenho comparando o agente de inspeção de pacotes instalado em uma máquina virtual utilizando *hypervisor* Xen[76] e o agente instalado diretamente na máquina *hypervisor* Xen, onde ele tem uma visão de todos os pacotes na mesma interface, aumentando assim seu desempenho. O objetivo do trabalho fica limitado ao uso em ambiente de nuvens e os testes realizados foram limitados à *switches* virtuais.

O SDNIPS[73] é um trabalho posterior dos mesmos autores do Snortflow que apresenta um sistema de prevenção de intrusão capaz de realizar ações de reconfiguração de rede. O agente Snort é instalado no domínio privilegiado da máquina virtual e interage com o *switch* OpenFlow virtual. Esse estudo, ao contrário do SnortFlow, mostra uma abordagem diferente, voltada para redes físicas e interagindo com o controlador OpenFlow, embora o trabalho proposto possa atuar sobre uma rede virtualizada, não teria o mesmo desempenho do SDNIPS.

Em 2014, foi proposto o BroFlow [21], que desenvolve uma arquitetura de detecção e prevenção de intrusão para Redes Definidas por Software. A criação de módulos de análise de tráfego que se comunicam com o controlador OpenFlow é interessante, pois tira o proveito de uma visão centralizada da rede na hora de tomar a decisão de bloqueio, assim como, o bloqueio pode ser feito mais próximo ao atacante. Porém o trabalho não tira vantagem das redes definidas por Software durante a análise do tráfego. Para fazer a análise, é necessário instalar um módulo chamado de sensor BroFlow em cada ponto da rede que se deseja analisar. Existem dois módulos: o de análise de redes virtuais e o de infraestrutura de rede.

O OpenSAFE [77] apresenta um sistema para redirecionar fluxos para aplicações que farão sua análise. A manipulação de fluxos do OpenSafe é realizada pelo uso do protocolo OpenFlow. Por meio dessa proposta, é possível aplicar filtros que selecionam um

fluxo específico e desvios para encaminhar o tráfego para as aplicações de monitoramento e segurança. Também existe um elemento no controlador OpenFlow que faz a interpretação das políticas e mensagens, definidas em uma linguagem de programação própria do OpenSafe, para assim, o controlador OpenFlow ser capaz de tomar decisões baseadas nessas informações.

Zanna et al. desenvolvem uma proposta de arquitetura para integração de um IDS em uma rede definida por software [78]. São realizados testes com clientes trocando pacotes através de um único *switch* OpenFlow e o desempenho da rede é avaliado em termos de como a monitoração afeta a latência e a vazão da rede. Os resultados obtidos são bons nesse cenário de teste simplificado.

O presente estudo, IDSFlow, contribui com alguns algoritmos para o controle dinâmico da análise de fluxos e de um histórico de ataques. Esse controle dinâmico evita o espelhamento e análise de todos fluxos pelo Snort, o que oferece um desempenho melhor e um uso mais racional da infraestrutura de rede.

Capítulo 5

A Proposta

Devido a mudança de paradigma que as SDNs trazem, diversas possibilidades surgem, assim como, novos problemas de segurança de rede. A utilização de IDSs tem se mostrado essencias para a segurança de redes, com isso, a proposta desse trabalho é a utilização de ambos conceitos, SDN e IDS, para assim atingir melhores resultados, trazendo maior segurança a rede de computadores.

O nome dado ao trabalho é IDSFlow, pois utiliza um IDS para auxiliar na criação dos fluxos do OpenFlow. O objetivo é manter uma rede de computadores segura, evitando ao máximo tráfego malicioso. A utilização da visão centralizada e a possibilidade de bloqueio em qualquer *switch* OpenFlow auxiliam esse objetivo. Analisar o histórico de ataques anteriores está dentro do escopo do IDSFlow, assim como, diferenciar os tipos de redes de computadores, como por exemplo, uma rede de usuários e uma rede de servidores. Com base nesses diferentes tipos de informações o IDSFlow é capaz de gerar regras diferenciadas para diversos cenários.

O presente estudo apresenta uma solução baseada na utilização de programas de código aberto apresentados anteriormente. Os programas estão apresentados na Tabela 5.1, eles foram escolhidos de acordo com os estudos feitos no Capítulo 2 e no Capítulo 3.

Tabela 5.1: Principais programas utilizados

Tipo	**Programa**
Controlador OpenFlow	Ryu
Emulador de rede	Mininet
Switch OpenFlow	CpQd
IDPS/NIDS	Snort

A proposta é desenvolver uma forma de integração entre redes definidas por *software* e o IPS. O IPS utilizado é o Snort, pois como o Snort é um programa escrito para redes tradicionais, o seu uso nas redes OpenFlow pode ser de grande valia para um controle automatizado da rede. Com o uso de um controlador centralizado, é possível aliar inteligência aos alertas gerados pelo Snort com o fim de atuar automaticamente na rede.

Um dos grandes diferenciais que as redes definidas por *softwares* em conjunto com o Snort podem trazer é que o bloqueio dos pacotes pode ser feitos de forma mais eficiente. Com as redes tradicionais, o bloqueio de pacotes é feito, em sua grande maioria, nos *firewalls* da rede, localizados na borda da rede. Com isso, existe muito tráfego malicioso na rede utilizando banda desnecessariamente em *switches* que não tem a capacidade de bloquear os mesmos. Com os *switches* OpenFlow e o IDSFlow, é possível bloquear o tráfego na porta onde ele é detectado, evitando assim uso de recursos desnecessariamente. Além disso, é possível alterar a prioridade de pacotes suspeitos, assim o tráfego suspeito e/ou malicioso terá menor impacto na rede de computadores.

Sendo assim, esse estudo é focado em desenvolver um modelo de integração entre controlador e IPS que seja eficiente e realizar experimentos que demonstrem a viabilidade dessa integração.

5.1 Arquitetura da proposta

A proposta visa solucionar problemas de análise de tráfego em cenários de redes como o apresentado na Figura 5.1. É possível observar que existem múltiplos *switches* OpenFlow e múltiplos Snorts. Basicamente, para cada *switch* do *core* presente na rede, tem-se um Snort interligado. Assim, será possível monitorar o *core* da rede com mais eficiência. É importante frisar que o bloqueio dos pacotes pode ser feito em qualquer *switch* que seja OpenFlow, ou seja, o bloqueio pode ser feito em um *switch* que o usuário está conectado diretamente.

Essa topologia supõe que o tráfego em cada *switch* de *core* é aproximadamente igual. É possível alterar a quantidade de Snorts, caso exista muito tráfego em alguma localização. Para isso, basta adicionar um Snort a esse *switch* ou ainda nos *switches* de acesso das redes. O mesmo se aplica caso o tráfego seja muito baixo. Nesse caso, se agrega o tráfego de dois *switches* de core em um único Snort.

A proposta é flexível o bastante para acomodar diversos cenários. O controlador é capaz de fazer a configuração dos *switches* OpenFlow de acordo com o número de Snorts na rede. Com o controlador OpenFlow, é possível até monitorar a utilização de banda e recomendar que o tráfego seja direcionado de um Snort para outro, se assim necessário.

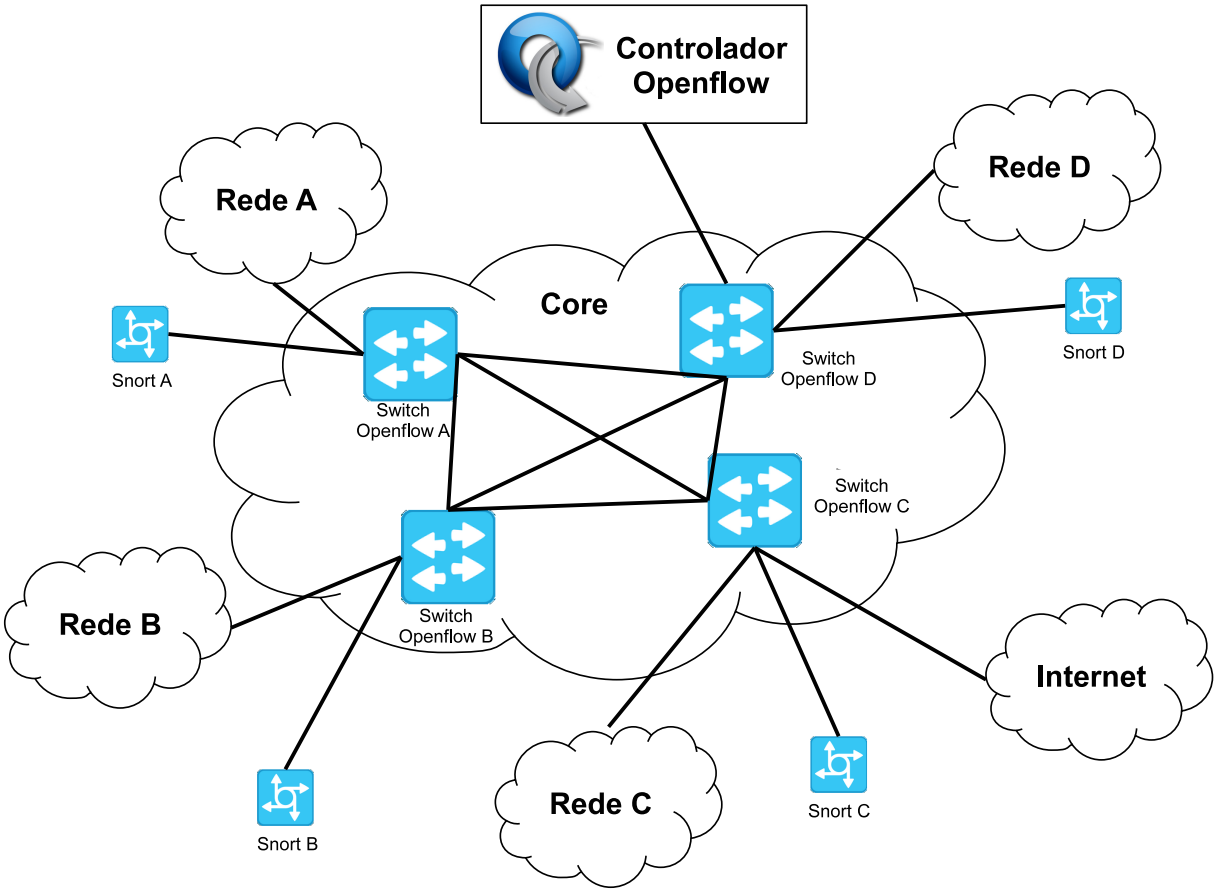

Figura 5.1: Visão da rede com a utilização do sistema proposto IDSFlow, que agrega a de detecção de intrusão e redes OpenFlow baseadas no controlador Ryu.

A arquitetura da solução IDSFlow pode ser vista na Figura 5.2, onde é possível observar alguns módulos dos componentes. Nessa figura, são mostrados os módulos que constituem o IDSFlow dentro do controlador Ryu, os quais interagem com o Snort através de um *pigrelay*.

5.1.1 Módulo *Pigrelay*

Especificamente, o *pigrelay* [79] é um software adicionado à máquina Snort, que verifica em tempo real os alertas gerados. A cada alerta gerado, o *pigrelay* envia ao controlador o pacote que gerou o alerta e a mensagem informativa sobre o alerta. O Snort chama esse conjunto de informações de *Alertpkt*.

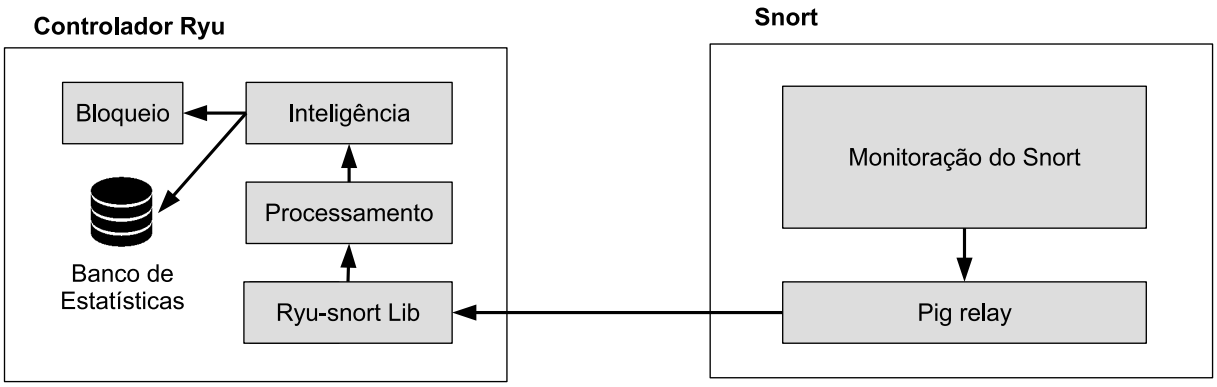

Figura 5.2: Arquitetura do sistema proposto, o IDSFlow.

Para o *pigrelay* ser capaz de enviar os alertas para o controlador, ele deve detectar quando um alerta é gerado. Para tal, o *pigrelay* fica escutando o *socket* que o Snort inicializa [80]. Essa é uma funcionalidade do Snort para que seja possível fazer tratamento dos pacotes que geraram os alertas por programas de terceiros. O Snort envia diversas informações sobre o pacote que gerou o evento de alerta. Dentre elas, o IDSFlow considera importante o datagrama original, além da mensagem do alerta configurado na regra do Snort. Todas essas informações são fornecidas para o controlador através do *Alertpkt* pelo *pigrelay*.O *Alertpkt* ainda contém as seguintes informações, ID do evento, libpcap [81] *pkthdr*, *offsets* para o *datalink*, *netlayer* e o cabeçalhos da camada de transporte.

Após a detecção do alerta, é necessário que o *pigrelay* seja capaz de enviar o datagrama e a mensagem do Snort para o controlador. Essa comunicação é por um *socket* de rede criado pelo *pigrelay*. Esse socket é usado para fazer uma conexão com o módulo Ryu-SnortLib, que será detalhado mais à frente.

Basicamente, o módulo *pigrelay* consiste em dois serviços: o primeiro é o serviço de comunicação com o controlador e o segundo é de leitura dos alertas do Snort.

Naturalmente, na comunicação entre o controlador e o Snort, é necessário que exista uma padronização para o envio dos dados. Especificamente, no IDSFlow, propõe-se que o Snort gere, minimamente, o seguinte conjunto de dados:

- idle_timeout: Tempo em que o fluxo do bloqueio fica ativo em caso de inatividade.
- hard_timeout: Tempo total em que o fluxo do bloqueio fica ativo independente da atividade.
- match_type: Qual a origem ou fluxo do ataque, que devem ser usados para o bloqueio. Essa origem/fluxo pode ser do tipo fonte apenas, fonte e destino, tipo de

tráfego de aplicação, ou qualquer conjunto de parâmetros que compõem a tupla de identificação de fluxos do OpenFlow.

- host: Identificador do Snort que está gerando a mensagem.

Esses parâmetros são enviados em um *JavaScript Object Notation* (JSON) e foram estruturados dessa maneira para garantir a modularidade, facilitando possíveis alterações e adição de novos parâmetros no futuro.

O IDSFlow busca alterar o mínimo possível o Snort, dando a possibilidade de integração e reutilização de regras de Snort já existentes. Assim, todo o legado de configuração de regras do Snort pode ser empregado de uma forma mais simples e fácil nas SDNs, uma vez que o necessário é apenas alterar a mensagem do alerta. A possibilidade de adicionar parâmetros à mensagem do Snort foi projetada de forma a dar mais autonomia ao administrador local, que poderá, assim, definir a criticidade dos alertas e qual ação deve ser tomada para garantir que um determinado pacote malicioso será enviado ao controlador. O controlador analisará essas informações e tomará uma decisão, que pode ser seguir a recomendação enviada pelo alerta do Snort ou apenas utilizar as informações em outro momento.

Além do lado do Snort, o IDSFlow compreende módulos no controlador Ryu, como mostrado na Figura 5.2. Basicamente, existem quatro módulos: Ryu-SnortLib, Processamento, Inteligência e Bloqueio.

5.1.2 Módulo Ryu-SnortLib

O módulo Ryu-SnortLib, é o responsável pelo recebimento do alerta do *pigrelay*. Para ser capaz de realizar o recebimento, o Ryu-Snortlib inicia um *socket* e fica escutando a rede por conexões. Essa conexão é utilizada pelo *pigrelay*, como explicado anteriormente. O *pigrelay* envia diversas informações para o módulo Ryu-SnortLib, que, por sua vez, faz a separação das informações enviadas referentes ao datagrama, à mensagem do Snort, dentre outras. Esse tratamento faz com que o controlador Ryu agora seja capaz de identificar as informações referentes ao pacote que gerou o alerta no Snort, como o *MAC Address*, endereço IP dentre outros atributos.

Após o tratamento da mensagem, o módulo Ryu-SnortLib gera um evento que é capturado pelo Módulo de Processamento. Esse evento contém diversas informações, provenientes do *Alertpkt* e da mensagem do Snort, porém no formato que o controlador

Ryu utiliza. Assim o controlador Ryu pode analisar o pacote que gerou o alerta caso necessário, além de identificar a mensagem enviada pelo Snort.

5.1.3 Módulo Processamento

O Módulo de Processamento faz a verificação da mensagem gerada pelo módulo Ryu-Snortlib. A mensagem do alerta contém vários parâmetros, tais como explicitados acima, referentes as informações que o administrador do Snort desejou enviar para o controlador Ryu.

Após o tratamento da mensagem, as informações são encaminhadas ao módulo de inteligência, que é o responsável por decidir a ação a ser tomada baseada nos parâmetros mostrados na mensagem e de outras informações. Algumas ações só serão tomadas em caso de reincidência do evento, ou serão aplicadas de forma mais abrangente. Informações como a de reincidência estão salvas no banco de dados do IDSFlow. Essas informações são de vital importância para a segurança da rede, pois são elas que proverão o necessário para o controlador tomar decisões mais complexas sobre o que bloquear e/ou analisar. Cabe também destacar que esse é um diferencial do IDSFlow, já que ele vai além da inteligência provida apenas pelo Snort.

5.1.4 Módulo de Inteligência

O módulo de inteligência é responsável pela análise do alerta e pelo envio dos comandos para o módulo de bloqueio.

Esse é o modulo capaz de utilizar a centralização da rede como um fator de grande benefício, pois com a visão completa da rede e com informações salvas no banco de dados, é possível tomar decisões mais inteligentes a respeito da rede. Por exemplo, é esse módulo que decide bloquear o envio de pacote diretamente no *switch* onde o tráfego está sendo gerado.

A tabela de fluxos dos *switches* é conhecida pelo módulo de inteligência. Com isso ele é capaz de escolher melhores regras para o bloqueio dos fluxos desejados. Esse módulo também é capaz de decidir sobre quais fluxos devem ser monitorados, além de adaptar automaticamente a frequência de análise pelo Snort, aumentando o desempenho da rede na detecção e bloqueio de intrusões. O detalhamento dos algoritmos desse módulo será feito na Seção 5.2.

5.1.5 Módulo de Bloqueio

O módulo de bloqueio é responsável por aplicar as regras de fluxo que foram determinadas pelo módulo de inteligência nos *switches* OpenFlow que estão relacionados ao fluxo que está sendo analisado.

Esse módulo é o responsável pela geração das regras de fluxo necessárias para os bloqueios e para a análise dos tráfegos. O módulo de bloqueio também verifica se é necessário atualizar alguma regra de bloqueio ou remover uma já configurada. Caso exista algum problema na geração do fluxo, ele avisa ao módulo de inteligência, como por exemplo, o *switch* não está mais acessível ou por algum motivo rejeitou a modificação da tabela de fluxos.

5.2 Algoritmos do módulo de inteligência

Para que o Snort possa analisar os pacotes, ele deve receber uma réplica dos mesmos. Para tanto, o IDSFlow cria um fluxo de envio das réplicas dos pacotes para o Snort, assim é possível que o Snort faça a analise do tráfego. É importante destacar que o IDSFlow não exige que o Snort funcione, em modo linha de rede, como uma ponte por onde todos os pacotes passam, mas pode redirecionar os tráfegos suspeitos para qualquer ponto da rede onde esteja o Snort.

A seleção dos fluxos de envio das réplicas é configurável. Isso é chamado de escopo de monitoração dentro do IDSFlow. O escopo de monitoração pode ser baseado em diversos parâmetros e essa proposta trata de basicamente dois escopos de monitoração: o primeiro baseia-se em monitorar todos os pacotes trafegados pela rede e o segundo em monitorar apenas uma porcentagem dos fluxos que o controlador adiciona aos *switches*. É possível verificar o fluxograma do processamento da configuração de um fluxo de envio nos *switches* OpenFlow na Figura 5.3. Já o diagrama de decisão na Figura 5.4 demonstra a forma de processamento de um alerta gerado pelo Snort.

A adaptação automática referente ao escopo de monitoração é feita no IDSFlow mantendo um histórico de endereços que já tenham sido infectados e configurando a esses endereços uma probabilidade maior de análise dado a quantidade de reincidência no período escolhido. O algoritmo simplificado utilizado para alterar o escopo de monitoração baseado nas portas de interesse e na reincidência dos alarmes é mostrado no Algoritmo 1.

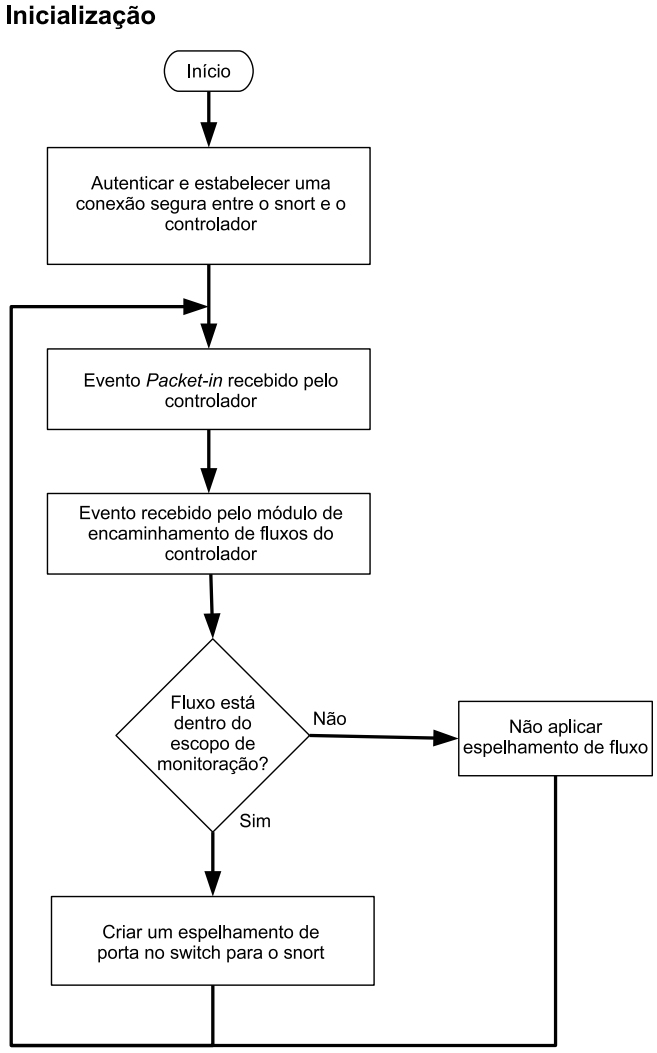

Figura 5.3: Processo de inicialização de fluxos.

Para cada novo fluxo, é verificado se o endereço de origem do equipamento de rede, já foi detectado anteriormente, em caso positivo, a probabilidade de análise do fluxo, aumenta de acordo com a quantidade de vezes que já foi detectado tráfego malicioso desse endereço de origem. Dessa forma, em caso de envio de tráfego malicioso, o endereço será bloqueado mais rapidamente, pois será analisado com mais frequência. Ao mesmo passo que, também verifica se a porta de destino requisitada é de interesse, em caso positivo, a probabilidade de analise aumenta, isso é importante quando o tráfego em certas portas tem uma probabilidade maior de tráfego malicioso, ou são mais importantes no ponto de vista do gerente da rede.

Para uma determinada definição de fluxo e uma porta de saída de um determinado *switch*, o algoritmo atualiza a probabilidade de analisar os fluxos mais granularizados ou não. Assim, se existe um alerta para fluxos na porta 80 passando em uma determinada porta de um *switch*, o algoritmo determinará qual o percentual dos fluxos que usem a porta

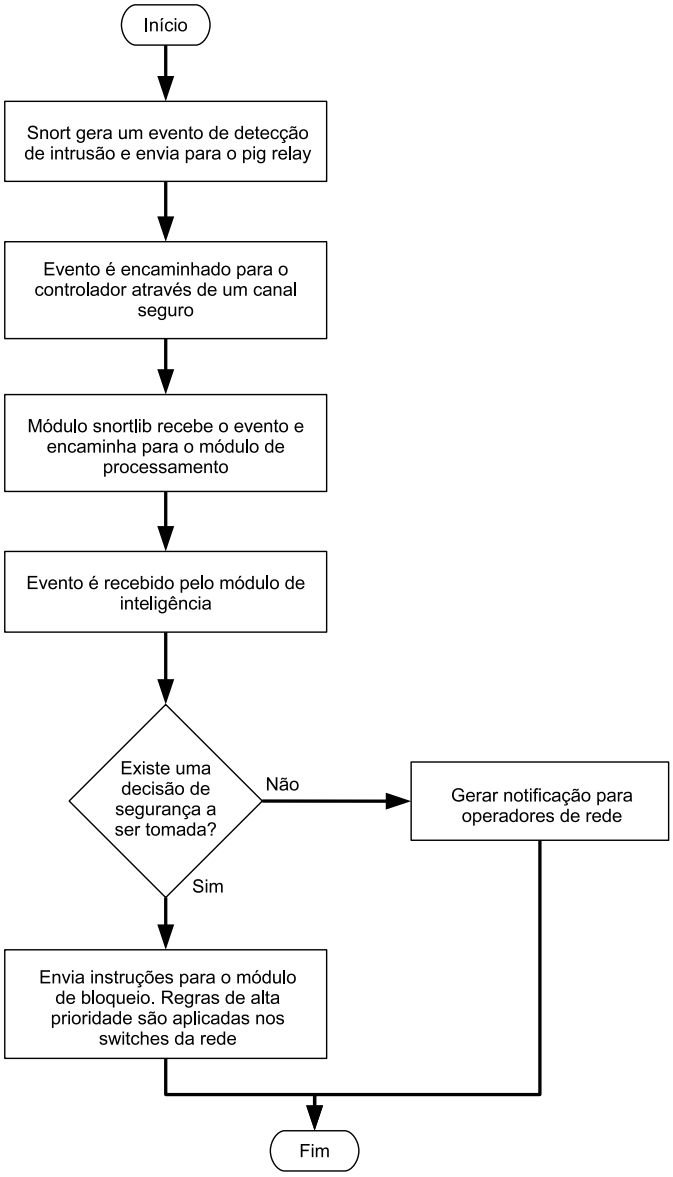

Figura 5.4: Processo de verificação do fluxo e instalação das ações corretivas.

80, considerando todos os campos do *match* OpenFlow, que deverão ser encaminhados para o Snort.

Algoritmo 1: Cálculo do peso da probabilidade de análise do fluxo, $P_{Analise}$.

input : $Peso_{reincidencia}$, $Peso_{portas}$, $P_{Analise}$, $definicao\_de\_fluxo, porta\_saida$;
output: $P_{Analise}$;

begin
  if *verifica_reincidência(definição_de_fluxo) == Verdadeiro* **then**
    $P_{Analise} = P_{Analise} \cdot (Peso_{reincidencia} \wedge reincidencias)$
  end
  if *checa_porta_de_interesse(porta_saida)* **then**
    $P_{Analise} = P_{Analise} \cdot Peso_{portas}$;
  end
end

Capítulo 6

Analise da Proposta

Visando analisar a proposta desde trabalho, foram executados diversos experimentos que tem como objetivo validar a proposta e analisar o seu desempenho. Os experimentos realizados foram elaborados para verificar se a análise realizada pelo Snort afeta o desempenho do controlador e também para apresentar formas diferenciadas de analisar os pacotes.

O primeiro experimento é baseado em aumentar a geração de alertas do Snort gradativamente e assim medir o tempo de resposta até a aplicação de um bloqueio de fluxo, ou seja o tempo desde que o controlador recebe a mensagem de alerta até o momento em que a ação contra o fluxo malicioso é posta em prática. Esse experimento verifica a capacidade de análise de grandes números de alertas do controlador.

O segundo experimento avalia formas de operação menos custosas para o Snort. Neste caso, ao invés de enviar todos os fluxos para o Snort, somente uma porcentagem dos fluxos é enviada. A efetividade do bloqueio foi comparada entre um espelhamento parcial e um espelhamento de todos os fluxos. Dessa forma, valida-se se o impacto de o controlador utilizar diferentes métodos para decidir o escopo de monitoração, dando maior flexibilidade à integração e tirando vantagem de um controle centralizado.

O terceiro experimento varia a quantidade de Snorts na rede para verificar a capacidade do controlador de comunicar-se com múltiplos Snorts. Nesse experimento, os Snorts enviam uma quantidade fixa de mensagens, diferente do primeiro experimento, em que o número de mensagens foi aumentando gradativamente. O objetivo desse experimento é verificar se o controlador está preparado para a topologia proposta nesse trabalho, pois ela conta com múltiplos Snorts enviando pacotes simultaneamente.

O quarto e o quinto experimento analisam a eficácia do algoritmo de alteração do escopo de monitoração que foi proposto (Algoritmo 1). Assim, analisou-se separadamente as duas características consideradas no algoritmo: a reincidência de ataques e a análise das portas de maior probabilidade de ataque.

O quarto experimento implementa a alteração do escopo de monitoração dado portas de interesse, ou seja, o administrador da rede escolhe quais portas são de maior interesse para os atacantes ou para o funcionamento da rede. Para cada novo fluxo analisado, será verificado se a porta do fluxo está no conjunto de portas de interesse e, em caso positivo, a probabilidade de análise será incrementada. O objetivo desse experimento é verificar o tráfego da rede que mais interessa ao administrador, evitando assim, o desperdício de recursos computacionais. Vale dizer que, para esse experimento, é suposto que as portas de interesse têm uma maior probabilidade de conter trafego malicioso. Um bom exemplo é uma rede em que só existam servidores de sites. Para esse caso, o administrador pode escolher uma probabilidade maior apenas para a porta 80, pois os ataques à rede que podem ter um impacto maior, provavelmente, serão nessa porta.

O quinto experimento, assim como o quarto, implementa a alteração do escopo de monitoração, porém de uma forma diferente: para toda detecção de trafego malicioso, o controlador salva em um banco de dados a máquina que enviou esse trafego. Com esse banco de dados, é possível que, para cada novo fluxo, se verifique se a máquina já foi previamente utilizada para enviar tráfego malicioso. Em caso verdadeiro, a probabilidade de análise desse fluxo será incrementada considerando um fator de reincidência e a quantidade de vezes em que foi detectada a reincidência de tráfego malicioso. O objetivo desse experimento é verificar a eficácia de se manter um histórico do perfil de cada máquina, pois existem diversas máquinas infectadas pela rede que não devem ter o mesmo tratamento de máquinas não infectadas.

Para a realização desses experimentos, foi emulada uma rede OpenFlow utilizando-se o Mininet [82]. O Mininet é uma ferramenta para emulação de redes, utilizada em SDN, que permite criar diferentes infraestruturas virtuais de rede, com máquinas, *switches*, controladores e enlaces. O Mininet permite a integração da rede virtual com máquinas físicas, que podem rodar diferentes sistemas. Os clientes emulados dentro do Mininet ficam restritos ao uso do sistema Linux.

6.1 Análise do tempo de resposta

A topologia do primeiro experimento consiste de uma rede com um controlador, um *switch*, um Snort e duas máquinas, como mostrado na Figura 6.1. Para efeito de garantir que o desempenho de um serviço não interferisse nos demais, optou-se por usar uma máquina virtual para execução do controlador, uma máquina virtual para emular a rede com um *switch* e uma máquina virtual que emula duas máquinas utilizando o Mininet e uma máquina virtual para executar o Snort.

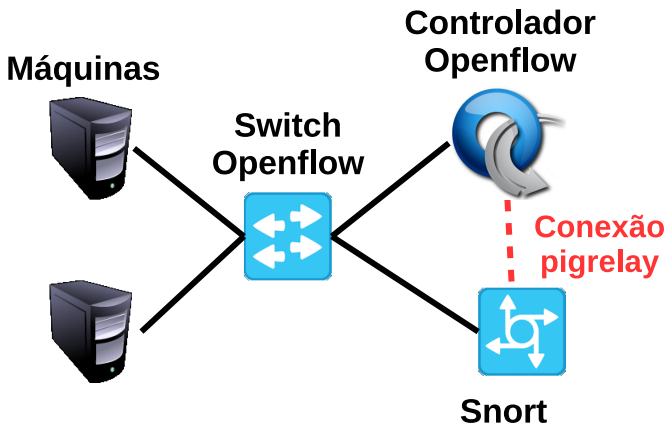

Figura 6.1: Topologia Experimento 1

Uma das máquinas da rede Mininet gera tráfego para a outra máquina, tráfego esse que possui características que levam o Snort a gerar alertas. A taxa de geração do tráfego é aumentada gradativamente a cada rodada, que por sua vez aumenta gradativamente os alertas gerados no Snort. Os valores dos parâmetros dos experimentos podem ser vistos na Tabela 6.1.

Tabela 6.1: Parâmetros do primeiro experimento.

Parâmetro	**Quantidade**
Controladores Ryu	1
Máquinas com Snorts	1
Máquinas com Mininet	1
Máquinas na rede Mininet	2
Repetições por iteração	20
Probabilidade padrão de análise	100%
Probabilidade de tráfego malicioso	100%
Incremento da taxa de pacotes maliciosos gerados por segundo	20
Número inicial e final de pacotes maliciosos enviados por segundo	20 e 300

O controlador foi propositalmente configurado para aplicar uma entrada de fluxo de

bloqueio que não afeta a geração dos alertas. Uma vez que desejamos medir o desempenho do controlador na presença de um número variado de alertas, o tráfego gerador dos alertas deve continuar inalterado para que o Snort envie as mensagens para o controlador. Sem essa modificação, o IDSFlow teria bloqueado o tráfego rapidamente e essa análise não seria possível.

O tempo medido foi o intervalo entre o recebimento do alerta do Snort pelo controlador e o envio da mensagem de adição de entrada do fluxo de bloqueio do controlador para o *switch* OpenFlow.

Foram realizados dois experimentos iguais em máquinas que hospedam o controlador com configurações diferentes. No primeiro, o controlador era uma máquina virtual com 512 MB de memória RAM e 1 processador virtual, enquanto que, no segundo, a memória RAM foi alterada para 1024 MB e 4 processadores virtuais. O objetivo era verificar como o controlador se comporta em máquinas de diferentes configurações.

A Figura 6.2 mostra os resultados encontrados.

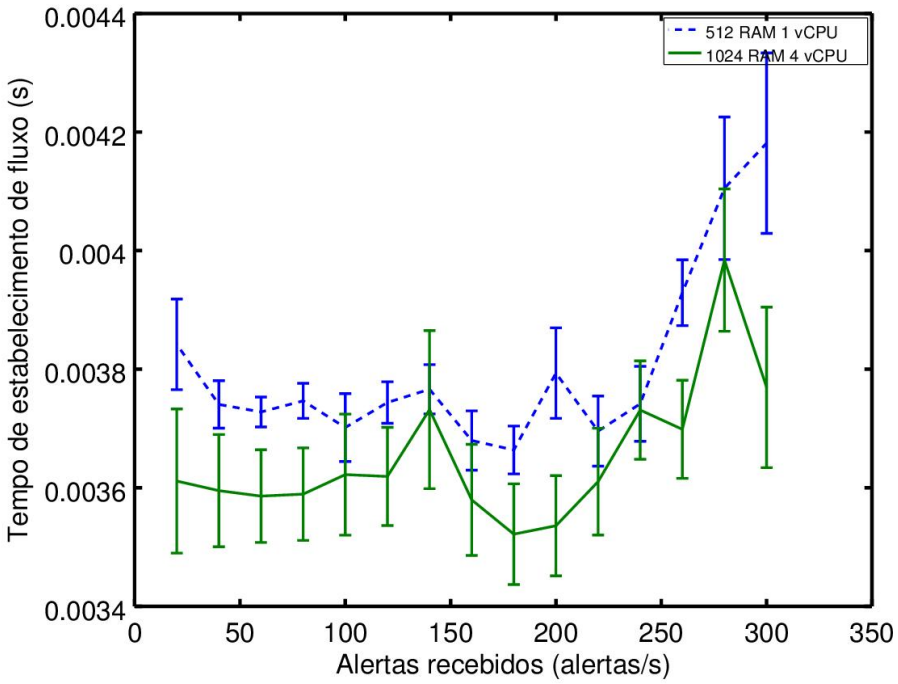

Figura 6.2: Tempo de estabelecimento de entrada de fluxo de bloqueio de acordo com o volume de alertas recebidos.

É possível observar que a partir de 200 alertas gerados por segundo, o tempo para aplicar o fluxo começa a aumentar substancialmente. Contudo, como os tempos são na ordem de milissegundos, pode-se dizer que o controlador tem um bom tempo de resposta

mesmo para uma taxa muito grande de alertas. Um detalhe importante é que o uso de memória pelo controlador é muito baixo: a utilização total de RAM, contando todos os programas na máquina, não ultrapassou 300 MB em nenhum momento do experimento. Contudo, a utilização de CPU aumentou com o número de alertas. Ao final do experimento com um processador na máquina do controlador, o uso do processador estava em 100%. No cenário do controlador com mais de um processador, alguns pequenos picos eram notados no segundo processador, enquanto o primeiro estava em 100%. Essas informações foram observadas durante o experimento utilizando o programa htop [83].

Nota-se que, para a configuração de máquina virtual com mais processadores e mais memória, existe uma pequena diferença nos tempos para o controlador aplicar o fluxo. Isso está de acordo com o esperado, pois, com mais processadores, é possível analisar e receber os alertas simultaneamente.

6.2 Análise da taxa de espelhamento de fluxos para o Snort

Para o segundo experimento, foi utilizado um cenário semelhante: uma máquina com o controlador Ryu, uma máquina para o Snort e uma máquina para executar o Mininet, que por sua vez, cria trinta máquinas em sua rede virtual. A topologia desse experimento pode ser verificada na Figura 6.3.

O experimento consiste em alterar a probabilidade de espelhar ou não um fluxo da rede e verificar a efetividade do Snort para cada uma das probabilidades avaliadas. A adaptação da taxa de espelhamento evita a saturação do enlace de dados para a máquina do Snort e também diminui a demanda de processamento da mesma.

Assim, ao invés de espelhar todos os pacotes para o Snort, o escopo de monitoração é modificado. Isso significa que cada fluxo gerado pelo controlador irá passar por uma verificação de conteúdo que determinará se o fluxo deve ser analisado. Nesse experimento, a probabilidade do fluxo estar no escopo de monitoração varia em passos de 0.1 na escala de 0 até 1.

Todas as trinta máquinas na rede virtual da Mininet geram tráfegos para uma das outras trinta máquinas sem repetir o mesmo destino. Nesse caso, são gerados trinta envios de pacotes pela rede virtual. Nesse conjunto de transmissão de pacotes, definiu-se que 10% do tráfego seria malicioso, ou seja, três máquinas irão enviar pacotes com conteúdo

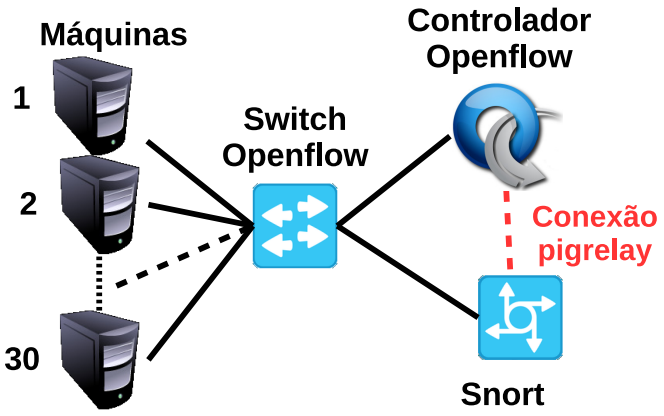

Figura 6.3: Topologia do experimento 2, que varia o percentual de fluxos que é espelhado para o Snort.

que irá gerar um alerta no Snort, que por sua vez, irá notificar o controlador Ryu. As máquinas que irão enviar esses fluxos maliciosos são aleatoriamente escolhidas.

Os valores dos parâmetros do experimentos podem ser vistos na Tabela 6.2

Tabela 6.2: Parâmetros do segundo experimento

Parâmetro	**Quantidade**
Controladores Ryu	1
Máquinas com Snorts	1
Máquinas com Mininet	1
Máquinas na rede Mininet	30
Repetições por iteração	40
Probabilidade inicial de análise	10%
Probabilidade final de análise	100%
Incremento da probabilidade por iteração	10%
Probabilidade de tráfego malicioso	10%

Nesse experimento, os módulos do IDSFlow no controlador de fato bloqueiam o fluxo que é considerado como malicioso, pois é necessário que o Snort pare de gerar alertas para o controlador após as ações serem tomadas na rede. Para cada rodada, foram medidas quantas vezes o controlador gerou um fluxo de bloqueio. Foram executadas quarenta rodadas para cada probabilidade de escopo de monitoração.

A Figura 6.4 mostra a efetividade dos bloqueios de acordo com a probabilidade do fluxo estar no escopo de monitoração.

De acordo com a Figura 6.4, é possível verificar que, utilizando uma simples probabilidade para alterar o escopo de monitoração, sua efetividade varia linearmente de acordo

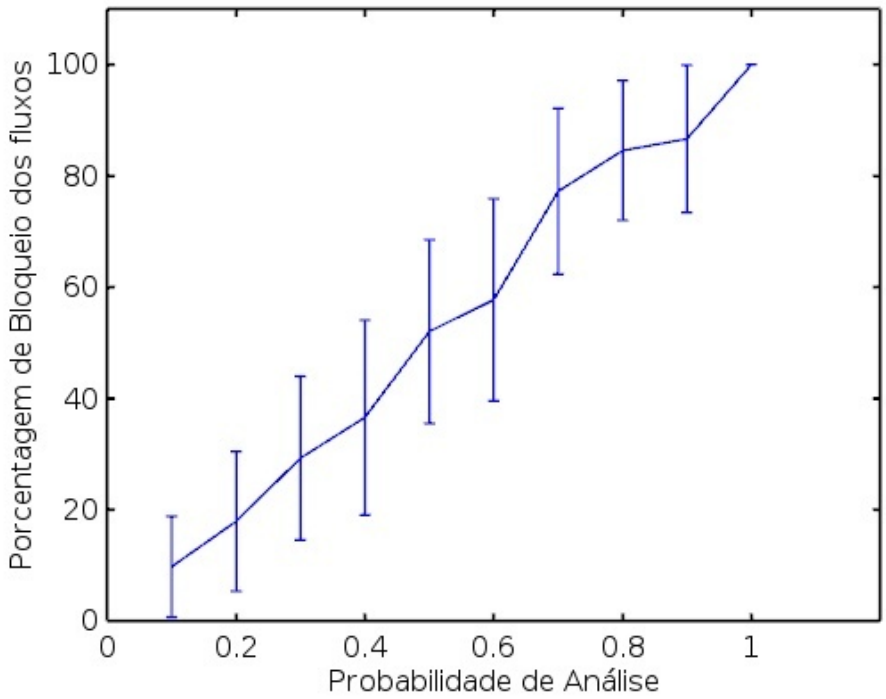

Figura 6.4: Percentual de bloqueios de fluxos maliciosos realizados de acordo com a probabilidade de espelhamento do fluxo para o Snort.

com a probabilidade escolhida. Esse resultado é esperado, pois se você analisa 10% dos fluxos da rede, se espera que detecte apenas 10% do tráfego malicioso. Essa é uma limitação natural do OpenFlow, já que não é possível fazer uma seleção aleatória dos pacotes em trânsito, mas apenas dos fluxos. Isso mostra a necessidade da adaptação inteligente da taxa de bloqueio proposta pelo IDSFlow com base em dados históricos, a qual não foi utilizada nesse teste propositalmente.

A configuração da máquina virtual usada com o Mininet era de 4 vCPUs e 1024 MB de memória RAM. Aumentar o número de máquinas, por consequência novos fluxos, iria dar maior confiabilidade aos resultados do experimento. No entanto, pode-se dizer que, embora a barra de erro esteja grande, ela está coerente com o resultado. De fato, para uma análise de todos os fluxos, não foram observados erros de classificação e todos os fluxos maliciosos foram bloqueados.

6.3 Análise da carga dos Snorts sobre o controlador

A arquitetura do terceiro experimento conta com uma máquina para ser o controlador RYU, uma máquina para executar o Mininet, que por sua vez cria duas máquinas em sua rede virtual, e de uma a dez máquinas executando o Snort. É possível verificar a topologia

do experimento na Figura 6.5.

A cada etapa do experimento o número de máquinas Snorts é incrementado. Aumentando o número de Snorts da rede é possível verificar a capacidade de escalar horizontalmente a análise de tráfego da rede.

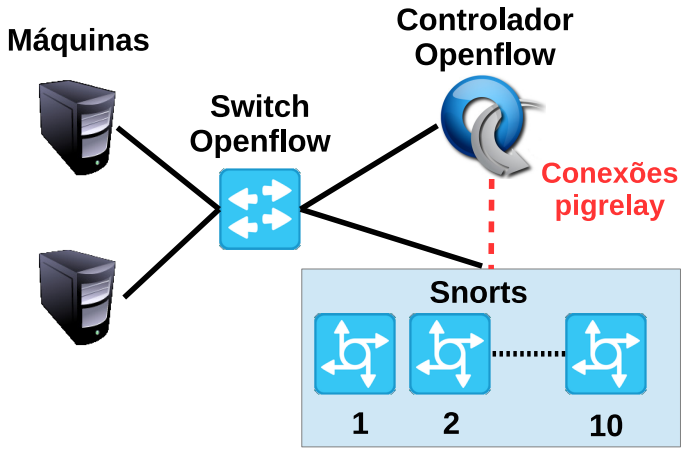

Figura 6.5: Topologia Experimento 3

Uma máquina da rede virtual do Mininet executa um *script* que gera tráfego malicioso, levando a alertas no Snort da mesma forma que no primeiro experimento. Nesse novo experimento, porém, a taxa dos alertas é mantida fixa em cada rodada. Apenas a quantidade de máquinas rodando o Snort é incrementada. Os Snorts recebem os mesmos pacotes maliciosos de toda a rede, pois o objetivo nesse experimento, é que cada Snort gere alertas em uma taxa igual para o controlador. O tempo medido foi o intervalo entre o controlador receber o alerta do Snort e adicionar o fluxo de bloqueio no *switch*. Os valores dos parâmetros do experimentos podem ser vistos na Tabela 6.3

Tabela 6.3: Parâmetros do terceiro experimento

Parâmetro	**Quantidade**
Controladores Ryu	1
Máquinas com Snorts inicial	1
Máquinas com Snorts final	10
Incremento de máquinas com Snort	1
Máquinas com Mininet	1
Máquinas na rede Mininet	2
Repetições por iteração	20
Probabilidade de analise	100%
Probabilidade de trafego malicioso	100%

A Figura 6.6 mostra os resultados encontrados. Novamente, os experimentos foram

executados em duas configurações de máquina diferentes para o controlador.

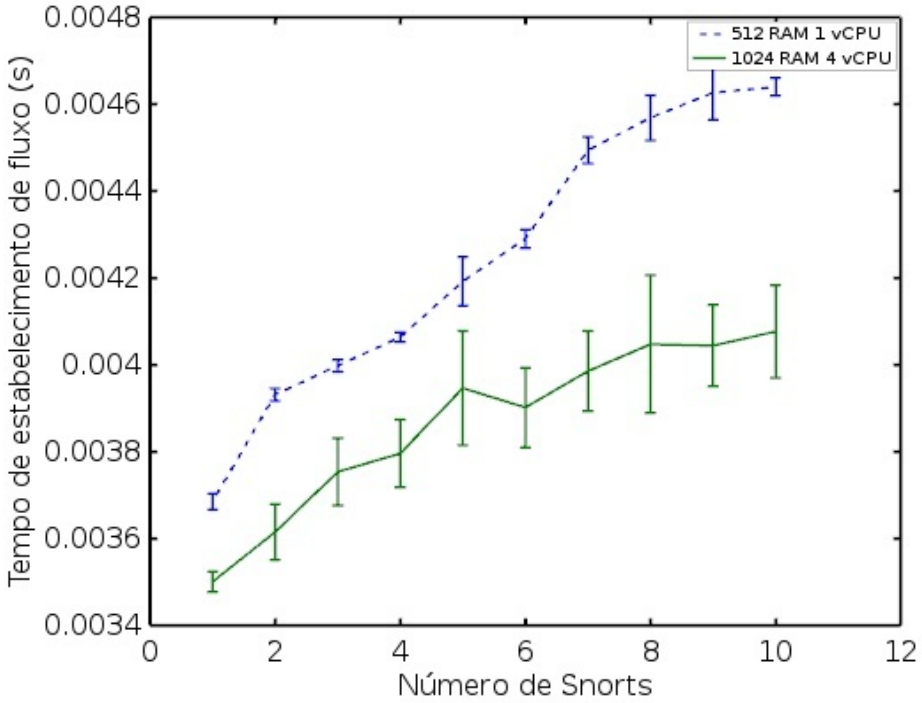

Figura 6.6: Tempo médio de estabelecimento dos fluxos de bloqueio de acordo com o número de Snorts na rede.

Analisando o gráfico, é possível verificar que conforme o número de Snorts aumenta e, por consequência, aumenta o número de alertas, o tempo de resposta do IDSFlow aumenta gradativamente. Também é possível verificar que, quando o controlador está na máquina de melhor configuração, o IDSFlow comporta-se ligeiramente melhor, conforme esperado. As observações referentes à utilização de memória e de CPU são iguais as do primeiro experimento.

6.4 Alteração do escopo por porta de interesse

Para o quarto experimento, o cenário utilizado é igual ao do segundo experimento: uma máquina com o controlador Ryu, uma máquina para o Snort e uma máquina para executar o Mininet, que por sua vez, cria 30 máquinas em sua rede virtual. A topologia do experimento está apresentada pela Figura 6.7.

O experimento tem como o objetivo verificar o quão melhor fica a taxa de bloqueio dado um incremento de probabilidade de análise devido a identificação das portas de interesse pelo administrador da rede. A cada etapa do experimento, o aumento da probabilidade devido a ser uma porta de interesse é incrementado, conforme estabelecido

pelo Algoritmo 1. Uma versão simplificada do algorítimo do controlador para esse caso específico pode ser visto no Algoritmo 2.

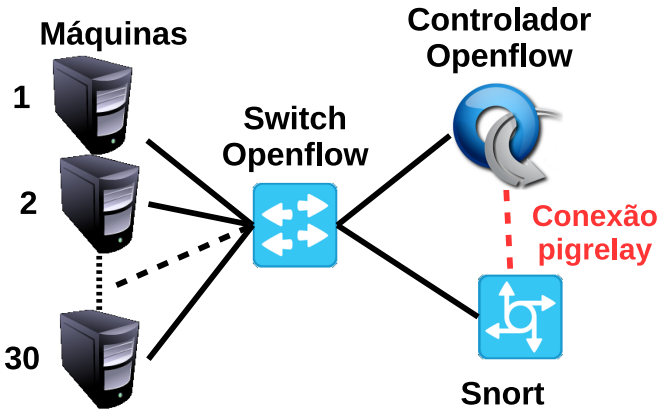

Figura 6.7: Topologia Experimento 4

A probabilidade de análise padrão do trafego é de 20% para o experimento e 50% do tráfego será em portas de interesse. Para o tráfego que está em portas de interesse, 30% é malicioso. Para o tráfego que está em portas comuns, apenas 10% é malicioso. Cada iteração do experimento será repetida 50 vezes. Os valores dos parâmetros do experimentos podem ser vistos na Tabela 6.4

Algoritmo 2: Alterando o escopo de monitoração por porta de interesse..

input : $Peso_{portas}$, $P_{Analise}$, $porta\_saida$;
output: $P_{Analise}$;

```
// Calculando o Peso da probabilidade de Analise do fluxo
```

begin
  if *checa_porta_de_interesse(porta_saida)* **then**
    $P_{Analise} = P_{Analise} \cdot Peso_{portas}$;
  end
end

A Figura 6.8 mostra os resultados encontrados.

É possível verificar que o número de bloqueio de pacotes aumenta em cada iteração, pois com o uso de um incremento maior da probabilidade de análise, dado que é uma porta de interesse, mais fluxos são analisados. Esse resultado era esperado, uma vez que, com uma maior probabilidade, mais pacotes serão enviados para o Snort, e, consequentemente, detectados. Esse experimento demonstra a capacidade do controlador de tomar decisões baseadas em novos fluxos de pacotes tendo, assim, um maior controle e segurança da rede de dados.

Tabela 6.4: Parâmetros do quarto experimento

Parâmetro	**Quantidade**
Controladores Ryu	1
Máquinas com Snorts	1
Máquinas com Mininet	1
Máquinas na rede Mininet	30
Repetições por iteração	50
Probabilidade padrão de análise	20%
Probabilidade de tráfego malicioso	10%
Probabilidade de tráfego malicioso quando porta de interesse	30%
Probabilidade de tráfego ser porta de interesse	50%
Incremento da probabilidade por iteração	10%
Incremento da probabilidade inicial até final	0% - 100%

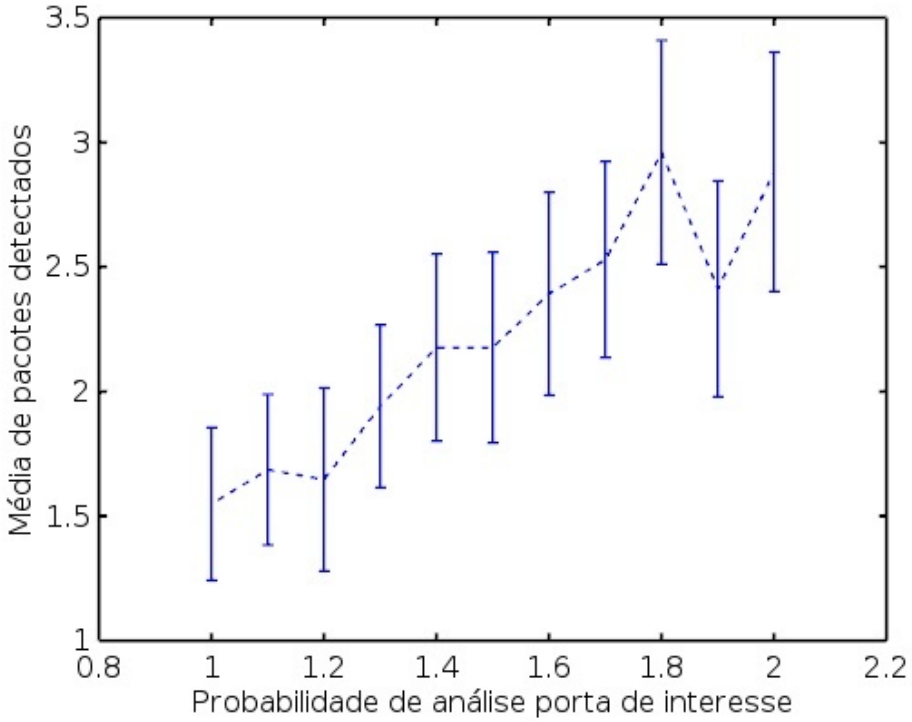

Figura 6.8: Número médio de pacotes detectados por iteração

6.5 Alteração do escopo por reincidência

A arquitetura do quinto experimento conta com uma máquina para ser o controlador RYU, uma máquina para executar o Snort e uma máquina para executar o Mininet, que por sua vez cria 30 máquinas em sua rede virtual. Pode-se verificar a topologia do experimento na Figura 6.9.

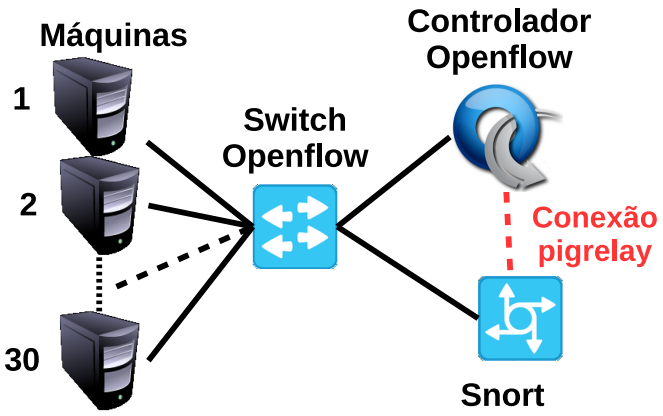

Figura 6.9: Topologia Experimento 5.

Para cada iteração, há um incremento de probabilidade de análise caso exista reincidência de detecção. Ao início de cada iteração, são definidas as máquinas que enviarão tráfego malicioso. Essas máquinas enviam um total de 30 fluxos com uma pausa entre cada fluxo, para que o *hard_ timeout* do *switch* seja atingido e o fluxo inicialmente configurado expire. Dessa forma, o próximo fluxo enviado irá passar pelo controlador RYU novamente, assim existe a reincidência do tráfego no controlador, mas não necessariamente ela será maliciosa.

O controlador por sua vez, salva a máquina em um banco de dados em caso de detecção de tráfego malicioso. Na próxima decisão de configuração de fluxo, o controlador verifica se essa máquina já enviou tráfego malicioso em seu banco de dados. Em caso positivo, a probabilidade de análise é aumentada, utilizando um multiplicador de acordo com o número de vezes que a máquina foi detectada. Dessa forma, se a máquina envia tráfego malicioso múltiplas vezes, o trafego vindo dela, será incrementada até tal ponto que todos os fluxos sejam analisados.

A fórmula para o cálculo da probabilidade de análise dada reincidência pode ser exemplificada como:

$$x = y(z)^n, \tag{6.1}$$

onde x é a probabilidade de análise final, y é a probabilidade de análise padrão, z é o

multiplicador dada a reincidência e n é a quantidade de vezes que a máquina se encontra no banco de dados, ou seja, quantas vezes já foi detectado tráfego malicioso proveniente da máquina.

O algoritmo descrito é uma versão simplificada do Algoritmo 1, como pode ser visto no Algorítimo 3.

Algoritmo 3: Alterando o escopo de monitoração por reincidência.

input : $Peso_{reincidencia}$, $P_{Analise}$, $definicao\_de\_fluxo$;
output: $P_{Analise}$;

```
// Calculando o Peso da probabilidade de Analise do fluxo
```

begin
  if *verifica_ reincidência(definição_ de_ fluxo)* $==$ *Verdadeiro* **then**
    $P_{Analise} = P_{Analise} \cdot (Peso_{reincidencia} \wedge reincidencias)$
  end
end

Esse experimento foi executado duas vezes com parâmetros de configuração do cenário diferentes, com o objetivo de verificar melhor a capacidade do algoritmo implementado.

A primeira parte foi executada com tráfego malicioso padrão de 20% do tráfego total, 25% de probabilidade de análise, 50 repetições por iteração, 19 reincidências por repetição e multiplicador sendo incrementado de 0.1 a cada iteração, variando de 1 a 2. Essa reincidências são a quantidade de vezes que um tráfego gerado foi enviado ao controlador, isso acontece devido a expiração do fluxo no *switch* OpenFlow.

A Tabela 6.5 apresenta os valores para essa primeira parte do experimento. A Figura 6.10 mostra os resultados encontrados.

Tabela 6.5: Parâmetros do quinto experimento parte 1

Parâmetro	**Quantidade**
Controladores Ryu	1
Máquinas com Snorts	1
Máquinas com Mininet	1
Máquinas na rede Mininet	30
Repetições por iteração	50
Reincidência por repetição	19
Probabilidade padrão de análise	25%
Probabilidade de tráfego malicioso	20%
Multiplicador de probabilidade de análise dada a reincidência	1 a 2
Incremento do multiplicador por iteração	0.1

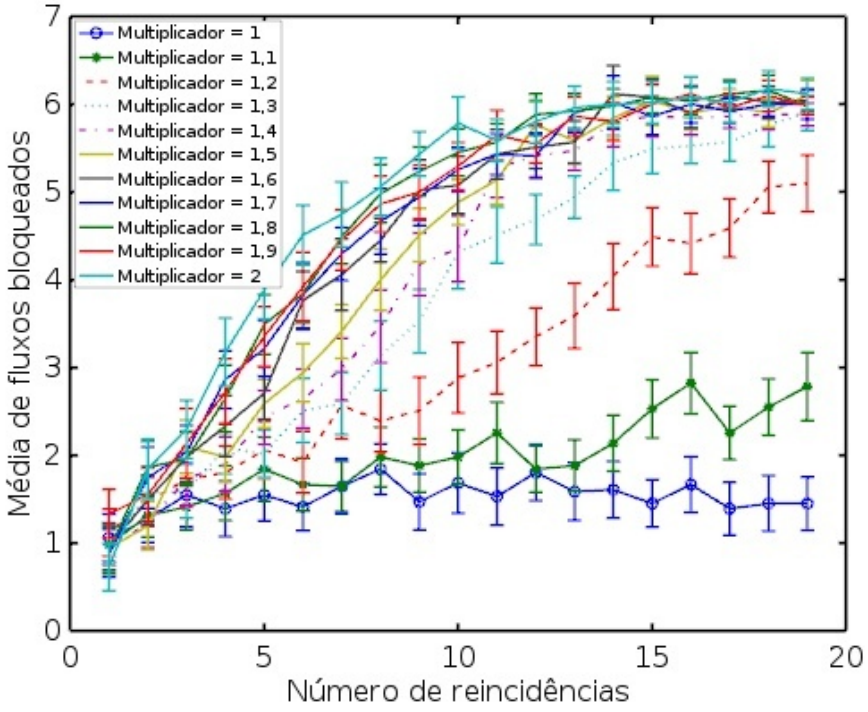

Figura 6.10: Número médio de fluxos bloqueados por reincidência.

Calculando o número de fluxos maliciosos para 30 máquinas, onde 20% das máquinas envia tráfego malicioso, existem seis máquinas que enviarão tráfego malicioso. Por uma análise da Figura 6.10, é possível ver que ao final do experimento, tem-se aproximadamente uma média seis pacotes sendo bloqueados ao final das reincidências para multiplicadores maiores que 1.2, ou seja, todos os fluxos maliciosos foram detectados.

Também é possível observar que após a sétima iteração do algoritmo sobre fluxo malicioso, as diferenças no bloqueio são muito pequenas. Isso é esperado, pois, de acordo com o cálculo da probabilidade de análise e seus parâmetros, os valores se tornam muito próximos a um conforme existe um número maior de reincidências detectadas.

A Tabela 6.6 mostra a probabilidade de análise dada a reincidência detectada. Vale lembrar que o número de reincidências não corresponde ao número de reincidências detectadas, pois a probabilidade de análise não é sempre um, portanto, não é todo pacote malicioso que irá gerar um alerta.

A segunda parte do experimento foi executada com tráfego malicioso padrão de 1% do tráfego total, 10% de probabilidade de análise, 50 repetições por iteração, 30 reincidências por repetição e multiplicador sendo incrementado de 1 a cada iteração, variando de 1 a 4. A Tabela 6.5 apresenta os valores para essa segunda parte do experimento. A Figura 6.11 mostra os resultados encontrados.

Tabela 6.6: Cálculo probabilidade de análise.

Probabilidade de Análise, assumindo probabilidade padrão=0,25							
Reincidência Detectada / Multiplicador	1	2	3	4	5	6	7
1	0,250	0,250	0,250	0,250	0,250	0,250	0,250
1.1	0,275	0,303	0,333	0,366	0,403	0,443	0,487
1.2	0,300	0,360	0,518	0,622	0,747	0,896	>1
1.3	0,325	0,423	0,549	0,714	0,928	>1	>1
1.4	0,350	0,490	0,686	0,960	>1	>1	>1
1.5	0,375	0,563	0,844	>1	>1	>1	>1
1.6	0,400	0,640	>1	>1	>1	>1	>1
1.7	0,425	0,723	>1	>1	>1	>1	>1
1.8	0,450	0,810	>1	>1	>1	>1	>1
1.9	0,475	0,903	>1	>1	>1	>1	>1
2	0,500	1	>1	>1	>1	>1	>1

Tabela 6.7: Parâmetros da segunda parte do quinto experimento.

Parâmetro	**Quantidade**
Controladores Ryu	1
Máquinas com Snorts	1
Máquinas com Mininet	1
Máquinas na rede Mininet	30
Repetições por iteração	30
Reincidência por repetição	49
Probabilidade padrão de analise	10%
Probabilidade de trafego malicioso	1%
Multiplicador de probabilidade de analise dada a reincidência	1 a 4
Incremento do Multiplicador por iteração	1

Calculando o número de fluxos maliciosos para 30 máquinas, onde 1% é tráfego malicioso, existe uma ou menos máquinas que enviarão tráfego malicioso. Em uma análise da Figura 6.11, é possível ver que, ao final do experimento, tem-se aproximadamente uma média de um pacote sendo bloqueado ao final das reincidências, devido a multiplicadores mais agressivos e ao maior número de reincidências, todos os pacotes são bloqueados ao final do experimento.

Também é possível observar que, após a trigésima iteração, as diferenças no bloqueio são muito pequenas. Isso se justifica pela fórmula do cálculo da reincidência. A Tabela 6.8 mostra a probabilidade de análise dada a reincidência. Vale lembrar novamente que o número de reincidências não corresponde ao número de reincidências detectadas, pois a

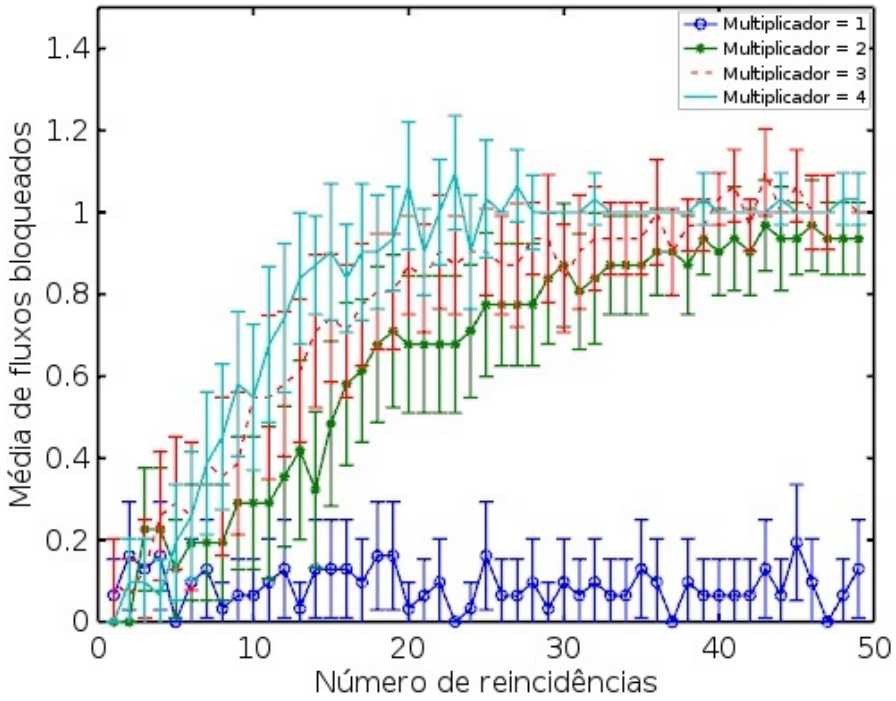

Figura 6.11: Número médio de fluxos bloqueados por reincidência.

probabilidade de analise não é sempre um.

Tabela 6.8: Calculo probabilidade de analise

Probabilidade de Analise, probabilidade padrão 0,1				
Multiplicador \ Reincidência Detectada	1	2	3	4
1	0,1	0,1	0,1	0,1
2	0,2	0,4	0,8	>1
3	0,3	0,9	>1	>1
4	0,4	>1	>1	>1

A segunda parte do experimento, mostra que, mesmo que a probabilidade de trafego malicioso seja baixa, é possível detectar o ofensor de forma rápida utilizando multiplicadores mais agressivos.

Foram feitos dois experimentos com o mesmo algorítimo para mostrar a flexibilidade do mesmo. É possível alterar os parâmetros de acordo com a rede a ser analisada.

Caso a máquina do Snort tenha processamento e rede limitadas, é possível utilizar uma probabilidade de análise inicial mais baixa, assim utilizando menos recursos da máquina. Contudo, será necessário ser mais agressivo uma vez que exista reincidência. Cada

operador de rede deve definir seus parâmetros de acordo com o tráfego na sua rede e com a capacidade de suas máquinas com o Snort.

Capítulo 7

Conclusão e Trabalhos futuros

7.1 Conclusões

De acordo com os estudos, é possível verificar a necessidade de segurança em SDNs. O IDSFlow torna as SDNs mais seguras, uma vez que, o sistema proposto viabiliza a aplicação de sistemas como o Snort com boa eficiência sem causar grande impacto no ambiente da rede OpenFlow.

Os algorítimos apresentados mostram a capacidade de integração entre os IDS e as SDNs, utilizando inteligência no controlador, é possível analisar fluxos mais prováveis de serem maliciosos. Trazendo assim um diferencial aos métodos comumente utilizados de análise de tráfego de redes. Com o IDSFlow, a análise do tráfego de redes ganha a visão da rede como um todo, provida pelo controlador OpenFlow, essa integração de IDSs com a visão da rede como um todo não existe nas redes de computadores atuais.

Os experimentos no ambiente emulado demonstraram a efetividade da proposta em diversos cenários. Por exemplo, no experimento onde o número de alertas é aumentado, demostra-se que o controlador é capaz de aguentar um grande número de alertas com uma configuração modesta de máquina. Vale ressaltar que o controlador, caso bem configurado, ao receber os alertas, efetivará o bloqueio do trafego malicioso, evitando assim, que o número de alertas continue aumentando. No segundo experimento, constatou-se que, é possível analisar apenas uma porcentagem da rede e conseguir uma taxa de efetividade de bloqueio referente a porcentagem de análise como observado na Figura 6.4. Por fim, no terceiro experimento é verificado que o controlador responde de acordo, caso exista a necessidade de múltiplos Snorts, como foi proposto nesse trabalho.

Os experimentos visavam validar a proposta feita inicialmente de acordo com a Fi-

gura 5.1, um único controlador e múltiplos Snorts. Também foi validada a situação onde seria inviável analisar todos os pacotes devido à capacidade limitada dos links das máquinas onde o Snort é executado.

Os dois experimentos finais, são mais focados em tirar proveito das SDNs, em vez de verificar a viabilidade do sistema proposto. O quarto experimento demonstra que é possível alterar a probabilidade de análise de um fluxo baseado em seus atributos. Por fim, no quinto experimento constata-se que é possível ter um histórico da rede e com isso ter melhores bloqueios e uso mais eficiente da rede. Com isso, o experimento demonstra a flexibilidade do sistema em diversos casos de uso.

Assim, por meio da análise da proposta e seus experimentos, demonstrou-se a viabilidade da integração de softwares já estabelecidos em redes de computadores tradicionais utilizados também em SDNs. Aumenta-se, assim, a segurança das SDNs, reutilizando os softwares e suas regras já funcionais.

A utilização de IDSs em conjunto das SDNs ainda pode melhorar muito, para uma melhora mais significativa, será necessário que o controlador se tenha um controle maior dos IDSs, como se o IDS fosse uma extensão do controlador. Os IDS tem que evoluir para poderem acompanhar as mudanças providas pelas SDNs, muitas regras utilizadas atualmente não devem funcionar em uma rede SDN devido as diferentes características dos fluxos de rede, uma vez que os pacotes percorrem rotas definidas pelo controlador que podem não corresponder aos protocolus usados atualmente.

7.2 Trabalhos Futuros

Existem muitas frentes para trabalhos futuros. Alguns tópicos são: algoritmos para alteração de escopo de monitoração baseado no fluxo, algoritmos para alteração da probabilidade de análise, utilização de múltiplos algoritmos alterando a probabilidade de análise, testes de comparação entre algoritmos, utilizar tráfegos reais, melhoria nas regras de detecção e alertas do Snort dentre outros.

A ideia de utilizar múltiplos algoritmos, como por exemplo o das portas de interesse e o de reincidência se mostrou interessante. No entanto, se deve levar em conta que, ao se utilizar múltiplos algoritmos, a probabilidade de análise pode sempre estar perto de 1, fazendo com que o uso de recursos não seja eficiente.

Sobre o uso eficiente de recursos de rede e computacionais, uma ideia é utilizar o

Snort de forma eficiente, ou seja, dependendo do tráfego enviado e da carga do Snort, seria possível alterar as probabilidades de análise para que o Snort receba de forma eficiente os pacotes a serem analisados.

Também é interessante que o controlador faça uma análise da utilização de banda e possivelmente das máquinas que executam os Snorts, podendo assim gerar alertas e recomendações para o administrador da rede. Recomendações como: a necessidade de utilização de um Snort para determinada parte de rede; realocação dos recursos do Snort; remoção do Snort dada baixa utilização; etc.

Uma comparação entre algoritmos e tipos de pacotes na rede se torna interessante, pois cria uma base para uso em casos reais, onde seria melhor empregado cada tipo de algoritmo. Por exemplo, em uma rede de servidores HTTP, é possível que o algoritmo de portas de interesse seja mais interessante do que o de reincidência. Já para redes de usuários, o de reincidência talvez seja mais eficiente. Utilizar novos algoritmos e comparar com tipos de tráfego facilitaria a utilização do sistema em redes de produção.

Dessa forma, é possível tirar proveito das SDNs, pois a centralização do controlador dá uma visão da rede como um todo, podendo assim tomar melhores decisões.

Bibliografia

[1] Symantec. *2015 Internet Security Threat Report*. Rel. téc. Abr. de 2015. URL: https://www.symantec.com/content/en/us/enterprise/other_resources/21347933_GA_RPT-internet-security-threat-report-volume-20-2015.pdf.

[2] G. O. S. Scott-Hayward e S. Sezer. "Sdn Security: A Survey". Em: *Future Networks and Services (SDN4FNS)* (2013), pp. 1–7. DOI: 10.1109/SDN4FNS.2013.6702553.

[3] N. McKeown, T. Anderson e H. Balakrishnan. "OpenFow:Enabling innovation in campus networks". Em: *ACM SIGCOMM Computer Communication Review* 38 (abr. de 2008), pp. 149–154. DOI: 10.1145/1355734.1355746.

[4] M. E. M. Campista e L. H. G. Ferraz. *Interconexão de Redes na Internet do Futuro: Desafios e Soluções*. Rel. téc. 2010. URL: http://www.gta.ufrj.br/ftp/gta/TechReports/CFM10.pdf.

[5] TrendMicro. *Ransomware*. URL: http://www.trendmicro.com/vinfo/us/security/definition/ransomware (acesso em 16/04/2016).

[6] S. Z. M. Shaid. "Malware behavior image for malware variant identification". Em: *Biometrics and Security Technologies (ISBAST), 2014 International Symposium on* (ago. de 2014), pp. 238–243. DOI: 10.1109/ISBAST.2014.7013128.

[7] M.-w. Wu. "A Stateful Approach to Spyware Detection and Removal". Em: *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)* (dez. de 2006), pp. 173–182. DOI: 10.1109/PRDC.2006.15.

[8] I. Ideses. "Adware detection and privacy control in mobile devices". Em: *Electrical and Electronics Engineers in Israel (IEEEI), 2014 IEEE 28th Convention of* (dez. de 2014), pp. 1–5. DOI: 10.1109/EEEI.2014.7005849.

[9] Y. Alkabani. "Trojan Immune Circuits Using Duality". Em: *Digital System Design (DSD), 2012 15th Euromicro Conference on* (set. de 2012), pp. 177–184. DOI: 10.1109/DSD.2012.134.

[10] A. Bhatia. "Java based Simulator to Detect Zero-Day Silent Worms using ACTM". Em: *Advance Computing Conference, 2009. IACC 2009. IEEE International* (mar. de 2009), pp. 847–852. DOI: 10.1109/IADCC.2009.4809125.

[11] F. Chen. "New Detection of Peer-to-Peer Controlled Bots on the Host". Em: *2009 5th International Conference on Wireless Communications, Networking and Mobile Computing* (set. de 2009), pp. 1–4. DOI: 10.1109/WICOM.2009.5302674.

[12] P. Gupta, C. Raissi e G. Dray. "SS-IDS: Statistical Signature Based IDS". Em: *Internet and Web Applications and Services, 2009. ICIW '09* (2009), pp. 407–412. DOI: 10.1109/ICIW.2009.67.

[13] H. Li e D. Liu. "Research on intelligent intrusion prevention system based on Snort". Em: *2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering* (2010), pp. 251–253. DOI: 10.1109/CMCE.2010.5610483.

[14] P. V. Amoli e T. Hämäläinen. "A real time unsupervised NIDS for detecting unknown and encrypted network attacks in high speed network". Em: *Measurements and Networking Proceedings (M&N), 2013 IEEE International Workshop on* (out. de 2013), pp. 149–154. DOI: 10.1109/IWMN.2013.6663794.

[15] S. Rubin, S. Jha e B. P. Miller. "Automatic generation and analysis of NIDS attacks". Em: *Computer Security Applications Conference, 2004. 20th Annual* (dez. de 2004), pp. 28–38. DOI: 10.1109/CSAC.2004.9.

[16] S. Vasanthi e S. Chandrasekar. "A study on network intrusion detection and prevention system current status and challenging issues". Em: *Advances in Recent Technologies in Communication and Computing (ARTCom 2011), 3rd International Conference on* (2011), pp. 181–183. DOI: 10.1049/ic.2011.0075.

[17] M. Washburn. *Apresentação IDS*. URL: https://www.cs.clemson.edu/course/cpsc420/presentations/Fall2010/IDS.ppt (acesso em 29/03/2016).

[18] R. Sommer e V. Paxson. "Enhancing byte-level network intrusion detection signatures with context". Em: *CCS '03 Proceedings of the 10th ACM conference on Computer and communications security* (out. de 2003), pp. 162–271. DOI: 10.1145/948109.948145.

[19] Crackers. *C99*. URL: https://www.aldeid.com/wiki/C99Shell (acesso em 16/02/2016).

[20] ietf. *HTTP*. URL: https://tools.ietf.org/html/rfc2616 (acesso em 06/01/2014).

[21] M. E. A. Lopez. "Uma Arquitetura de Detecção e Prevenção de Intrusão para Redes Definidas por Software". Em: (maio de 2014).

[22] N. Desai. *IPS next step in the evolution of IDS*. URL: http://www.symantec.com/connect/articles/intrusion-prevention-systems-next-step-evolution-ids (acesso em 16/02/2016).

[23] L. Hui e C. Yonghui. "Research Intrusion Detection Techniques from the Perspective of Machine Learning". Em: *Multimedia and Information Technology (MMIT), 2010 Second International Conference on* (2010), pp. 166–168. DOI: 10.1109/MMIT.2010.161.

[24] I. Ahmad, A. B. Abdullah e A. S. Alghamdi. "Application of artificial neural network in detection of probing attacks". Em: *Industrial Electronics and Applications, 2009. ISIEA 2009. IEEE Symposium on* (2009), pp. 557–562. DOI: 10.1109/ISIEA.2009.5356382.

[25] S. Murugan e K. Kuppusamy. "System and methodology for unknown Malware attack". Em: *Sustainable Energy and Intelligent Systems (SEISCON 2011), International Conference on* (2011), pp. 803–804. DOI: 10.1049/cp.2011.0475.

[26] M. Roesch. *Snort Webpage*. URL: https://www.snort.org/ (acesso em 29/03/2016).

[27] T. Xing et al. "SnortFlow: A OpenFlow-Based Intrusion Prevention System in Cloud Environment". Em: *Research and Educational Experiment Workshop (GREE), 2013 Second GENI*. Mar. de 2013, pp. 89–92. DOI: 10.1109/GREE.2013.25.

[28] the Open Information Security Foundation. *Suricata*. URL: https://suricata-ids.org/ (acesso em 16/02/2016).

[29] V. Paxson. "Bro: a system for detecting network intruders in real-time". Em: *Computer Networks: The International Journal of Computer and Telecommunications Networking* (dez. de 199), pp. 2435–2463. DOI: 10.1016/S1389-1286(99)00112-7.

[30] NVIDIA. *CUDA*. URL: http://www.nvidia.com.br/object/cuda_home_new_br.html (acesso em 29/03/2016).

[31] Y. Liu. "Innovative approach for porting existing CPU program to its CUDA program". Em: *Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on* (nov. de 2015), pp. 1503–1508. DOI: 10.1109/BIBM.2015.7359898.

[32] NVIDIA. *NVIDIA*. URL: http://www.nvidia.com.br/ (acesso em 29/03/2016).

[33] L. G. Roberts. "The evolution of packet switching". Em: *Proceedings of the IEEE (Volume:66 , Issue: 11)* (nov. de 1978), pp. 1307–1313. DOI: 10.1109/PROC.1978.11141.

[34] M. Casado, T. Garfinkel e A. Akella. "SANE: A Protection Architecture for Enterprise Networks". Em: *USENIX-SS'06 Proceedings of the 15th conference on USENIX Security Symposium - Volume 15 Article No. 10* (2006).

[35] M. Casado, M. J. Freedman e J. Pettit. "Ethane: taking control of the enterprise". Em: *SIGCOMM '07 Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications* (ago. de 2007), pp. 1–12. DOI: 10.1145/1282380.1282382.

[36] O. N. Foundation. *SDN*. URL: https://www.opennetworking.org/sdn-resources/sdn-definition (acesso em 29/03/2016).

[37] O. N. Foundation. *Open Networking Foundation*. URL: https://www.opennetworking.org/ (acesso em 29/03/2016).

[38] T. D. Nadeau e K. Gray. *SDN: Software Defined Networks*. O'Reilly Media; 1 edition (September 7, 2013), 2013.

[39] H. E. Egilmez. "Scalable video streaming over OpenFlow networks: An optimization framework for QoS routing". Em: *2011 18th IEEE International Conference on Image Processing* (set. de 2011), pp. 2241–2244. ISSN: 1522-4880. DOI: 10.1109/ICIP.2011.6116083.

[40] N. McKeown. *How SDNs will tame networks*. URL: http://yuba.stanford.edu/~nickm/talks/Hot%20Interconnects%20Keynote%202012%20POSTED.pptx (acesso em 29/03/2016).

[41] O. N. Foundation. *OpenFlow*. URL: https://www.opennetworking.org/sdn-resources/openflow (acesso em 29/03/2016).

[42] Y. Watashiba, S. Hirabara e S. Date. "OpenFlow Network Visualization Software with Flow Control Interface". Em: *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual* (jul. de 2013), pp. 475–477. DOI: 10.1109/COMPSAC.2013.81.

[43] OpenFlow. *OpenFlow Releases Archive*. URL: http://archive.openflow.org/wk/index.php/OpenFlow_Releases (acesso em 16/02/2016).

[44] O. N. Foundation. *Release OpenFlow 1.0*. URL: http://archive.openflow.org/wk/index.php/OpenFlow_1.0_release_note (acesso em 16/02/2016).

[45] O. N. Foundation. *Release OpenFlow 1.1*. URL: http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf (acesso em 16/02/2016).

[46] O. N. Foundation. *Release OpenFlow 1.2*. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdff (acesso em 16/02/2016).

[47] O. N. Foundation. *Release OpenFlow 1.3*. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf (acesso em 16/02/2016).

[48] O. N. Foundation. *Release OpenFlow 1.4*. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf (acesso em 16/02/2016).

[49] O. N. Foundation. *Release OpenFlow 1.5*. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf (acesso em 16/02/2016).

[50] O. N. Foundation. *The Benefits of Multiple Flow Tables and TTPs*. Rel. téc. Fev. de 2015. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_Multiple_Flow_Tables_and_TTPs.pdf.

[51] C. E. Rothenberg, M. R. Nascimento e M. R. Salvador. "OpenFlow e redes definidas por software : um novo paradigma de controle e inovação em redes de pacotes". Em: *Control, vol. 7* (2011), pp. 65–75.

[52] N. Gude, T. Koponem e J. Pettit. "NOX towards an operating system for networks". Em: *ACM SIGCOMM Computer Communication Review* (jul. de 2008), pp. 105–110. DOI: 10.1145/1384609.1384625.

[53] N. repo. *NOX Github*. URL: https://github.com/noxrepo/nox (acesso em 15/04/2016).

[54] N. Repo. *POX*. URL: http://www.noxrepo.org/ (acesso em 15/04/2016).

[55] N. repo. *POX Github*. URL: https://github.com/noxrepo/pox (acesso em 15/04/2016).

[56] Floodlight. *Floodlight controller*. URL: http://www.projectfloodlight.org (acesso em 29/03/2016).

[57] S. Microsystems. *Java*. URL: https://www.java.com (acesso em 29/03/2016).

[58] Floodlight. *Floodlight Github*. URL: https://github.com/floodlight/floodlight (acesso em 29/03/2016).

[59] Osrg. *Ryu SDN controller*. URL: https://osrg.github.io/ryu/ (acesso em 29/03/2016).

[60] M. Casado. *Lista de OpenFlow Software*. URL: http://yuba.stanford.edu/~casado/of-sw.html (acesso em 16/02/2016).

[61] O. Vswitch. *OVS*. URL: http://openvswitch.org/ (acesso em 15/02/2016).

[62] D. Ritchie. *C*. URL: http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1256.pdf (acesso em 15/02/2016).

[63] O. Vswitch. *FAQ OVS*. URL: https://github.com/openvswitch/ovs/blob/master/FAQ.md#q-does-open-vswitch-support-openflow-meters (acesso em 23/02/2016).

[64] CpQd. *CpQd Software Switch*. URL: http://cpqd.github.io/ofsoftswitch13/ (acesso em 23/02/2016).

[65] Z. L. Kis. *Openflow 1.1 SoftSwitch*. URL: https://openflow.stanford.edu/display/of11softswitch/Home (acesso em 23/02/2016).

[66] Ryu. *Ryu Certification*. URL: https://osrg.github.io/ryu/certification.html (acesso em 23/02/2016).

[67] P. g. National Science Foundation. *ns-3*. URL: https://www.nsnam.org (acesso em 29/03/2016).

[68] S.-Y. Wang, C.-L. Chou e C.-M. Yang. "EstiNet 8.0 OpenFlow Network Simulator and Emulator". Em: *IEEE Communication Magazine, Vol. 51, Issue 9* (2013). DOI: 10.1109/MCOM.2013.6588659.

[69] B. Lantz e B. Heller. *Mininet*. URL: https://www.nsnam.org (acesso em 29/03/2016).

[70] K. Benton, L. J. Camp e C. Small. "OpenFlow Vulnerability Assessment". Em: *in Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013* (ago. de 2013), pp. 151–152. DOI: 10.1145/2491185.2491222.

[71] R. Kloti, V. Kotronis e P. Smith. "OpenFlow: A security analysis". Em: *Network Protocols (ICNP), 2013 21st IEEE International Conference on*. Out. de 2013, pp. 1–6. DOI: 10.1109/ICNP.2013.6733671.

[72] A. Lara, A. Kolasani e B. Ramamurthy. "Network Innovation using OpenFlow: A Survey". Em: *Communications Surveys Tutorials, IEEE* 16.1 (fev. de 2014), pp. 493–512. ISSN: 1553-877X. DOI: 10.1109/SURV.2013.081313.00105.

[73] T. Xing et al. "SDNIPS: Enabling Software-Defined Networking based intrusion prevention system in clouds". Em: *Proceedings of the 10th International Conference on Network and Service Management, CNSM 2014* (2015), pp. 308–311. DOI: 10.1109/CNSM.2014.7014181.

[74] Z. Hu, M. Wang e X. Yan. "A comprehensive security architecture for SDN". Em: *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on* (fev. de 2014), pp. 30–37. DOI: 10.1109/ICIN.2015.7073803.

[75] F. Nagahama, F. Farias e E. Aguiar. "IPSFlow – uma proposta de IPS distribuído para captura e bloqueio seletivo de tráfego malicioso em redes definidas por software". Em: *SBSeg 12* (nov. de 2012).

[76] J. Chen, R. Yang e J. Yin. "Improvements in Upgrading Xen-Virtualized Large Computer Systems". Em: (ago. de 2014), pp. 265–269. DOI: 10.1109/ES.2014.29.

[77] J. R. Ballard, I. Rae e A. Akella. "Extensible and scalable network monitoring using OpenSAFE". Em: *Proceeding INM/WREN'10 Proceedings of the 2010 internet network management conference on Research on enterprise networking* (abr. de 2010), pp. 8–8.

[78] P. Zanna et al. "Adaptive threat management through the integration of IDS into Software Defined Networks". Em: *Network of the Future (NOF), 2014 International Conference*. Dez. de 2014, pp. 1–5. DOI: 10.1109/NOF.2014.7119792.

[79] J. Lin. *Github Pigrelay*. URL: https://github.com/John-Lin/pigrelay (acesso em 29/03/2016).

[80] Snort. *Snort Socket*. URL: https://www.snort.org/faq/readme-unsock (acesso em 16/04/2016).

[81] tcpdump.org. *Libpcap*. 2000. URL: http://www.tcpdump.org/ (acesso em 06/01/2014).

[82] B. Heller. "Reproducible Network Research with High-Fidelity Emulation (Ph.D Thesis)". Em: June (2013).

[83] H. Muhammad. *Htop*. URL: http://hisham.hm/htop/ (acesso em 29/03/2016).