Uma abordagem autonômica para mitigar ciberataques em redes de computadores

Luiz Arthur Feitosa dos Santos

Tese apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Doutor em Ciências

Programa: Pós-Graduação em Ciência da Computação

Orientador: Prof. Dr. Daniel Macêdo Batista

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da Fundação Araucária

São Paulo, agosto de 2016

Uma abordagem autonômica para mitigar ciberataques em redes de computadores

Esta é a versão original da tese elaborada pelo candidato (Luiz Arthur Feitosa dos Santos), tal como submetida à Comissão Julgadora.

Resumo

SANTOS, Luiz A. F. **Uma abordagem autonômica para mitigar ciberataques em redes de computadores**. 2016. x f. Tese (Doutorado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2016.

Nos últimos anos observa-se o crescimento dos problemas relacionados com segurança em redes de computadores locais, que frequentemente são alvos ou fontes de diversos tipos de ciberataques. Em parte, isso ocorre porque as redes locais estão se tornando extremamente dinâmicas e heterogêneas. Como agravante, os ataques à segurança estão mais sofisticados, pois muitos são compostos por várias etapas e utilizam diferentes métodos para concretizar a investida, o que dificulta a identificação e reação contra essas ameaças. Desta maneira, manter a segurança em ambientes de rede tão heterogêneos e dinâmicos que são frequentemente expostos a pragas digitais, torna-se uma tarefa complexa para o administrador de rede. Nesse contexto, o presente trabalho tem por objetivo desenvolver uma arquitetura autonômica que mantenha a segurança de redes de computadores, exigindo o mínimo de intervenção humana. Para alcançar esse objetivo, propõem-se uma abordagem que emprega aprendizagem de máquina para processar, similarmente à memória humana, históricos de uso da rede e alertas de segurança, para extrair regras de segurança que são aplicadas autonomicamente na rede, por intermédio da tecnologia OpenFlow. Tal arquitetura, ainda propõem utilizar mensagens postadas em redes sociais, para extrair alertas de cibersegurança que auxiliem o administrador da rede a elucidar problemas na rede local. Nos experimentos executados, a arquitetura autonômica proposta conseguiu mitigar até 97,5% dos pacotes maliciosos referentes a ataques DDoS. Em experimentos com ataques DoS a metodologia conseguiu reduzir em até 99,95% a quantidade de pacotes maliciosos, o que atenuou os efeitos dos ataques nos demais fluxos de rede (não relacionados com os ataques), que chegaram a apresentar um aumento de até 752,42% na quantidade de pacotes transmitidos, quando comparados a fluxos de redes sem proteção e sob o mesmo ataque. Em experimentos em uma rede real em produção a arquitetura ajudou a conter 44 tipos de ciberameaças, sem causar distúrbios na rede. Quanto aos experimentos com extração de alertas a partir de mensagens em redes sociais, constatou-se que mais de 50% dos alertas de segurança extraídos pela metodologia proposta representam efetivamente ciberameaças. Portanto, conclui-se que a arquitetura autonômica proposta consegue prover meios para que as redes de computadores criem regras de segurança, fundamentadas em históricos de uso da rede, de forma a mitigar ciberameaças que estejam influenciando negativamente a rede. Também, conclui-se que é possível extrair alertas de cibersegurança de mídias não tradicionais, tal como as redes sociais. O que pode colaborar na identificação rápida de novas ameaças que estejam afetando as redes de computadores mundo a fora.

Palavras-chave: Redes Definidas por Software, Computação Autonômica, Segurança.

Abstract

SANTOS, Luiz A. F. **An autonomic approach to mitigate cyberattacks in computer networks** . 2016. x f. Tese(Doutorado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2016.

Nowadays it is possible to observe the growth of computer security problems, mainly the threats related with local computer networks, which are often targets or sources of many types of security attacks. One reason for this is that local networks are becoming extremely dynamic and heterogeneous. To aggravate this situation, the security threats are becoming more sophisticated too. For instance, many attacks use several steps and different methods to achieve their objective, which usually complicates the identification and reaction against these threats. In this scenario, it is a hard task for a human to deal with cyber attacks, mainly due the increasing number of users and heterogeneous devices in LAN encouraged by practices such as BYOD, that constantly brings new threats to these environments. Therefore, to reduce the necessity of human interaction to maintain the network security, we propose an autonomic approach that uses machine learning to process, in a similar way of human memory, the historical of network usage and security alerts, to generate security rules, that are imposed to network using SDN resources, to mitigate cyber attacks. In addition, we propose a method of extracting cyber alerts based on messages posted on social networks, which can be used to prevent security problems in computing networks. During the experiments, the autonomic architecture proposed was able to mitigate 97.5% of malicious packets generated by DDoS attacks. In DoS experiments, the methodology was able to reduce 99.95% of malicious packets and attenuated the negative effect of the attack, increasing 752.42% the amount of packets sent/received in network flows not related with attacks. In a real network, the architecture mitigated 44 types of cyber threats, without disturbing the network. In the experiments with social network messages, was observed that more than 50% of alerts generated by our methodology represent effectively threats. Therefore, we conclude that the autonomic architecture proposed is able to create security rules to prevent and protect computer networks of cyber attacks. Also, was observed that it is possible to extract cyber alerts from messages posted in social network. Thus, social networks are a new source of data that can be used to prevent or elucidate new security problems in computer networks.

Keywords: Software-defined networking, Autonomic Computing, Security.

Sumário

Lista de Abreviaturas

ACSM	*Autonomic Cyber Security Management*
AN	*Active Networks*
API	*Application Programming Interface*
ARP	*Address Resolution Protocol*
ASM	*Autonomic Security Management*
Auto-AID	*Autonomic Anomaly IDentification*
AVPS	*Autonomic Violation Prevention System*
Bps	*Bytes* por segundo
BYOD	*Bring Your Own Device*
BYOT	*Bring Your Own Threat*
CA	Computação Autonômica
CAIS	Centro de Atendimento a Incidentes de Segurança
CGI	*Common Gateway Interface*
CIFS	*Common Internet File System*
CPU	*Central Processing Unit*
CUSUM	*Cumulative Sum*
D-CAF	*Distributed Context-Aware Firewall*
DDoS	*Distributed Denial of Service*
DNS	*Domain Name System*
DoS	*Denial of Service*
DXCCDF	*Distributed eXtensible Configuration Checklist Description Format*
ECA	Evento-Condição-Ação
ECN	*Explicit Congestion Notification*
ForCES	*Forwarding and Control Element Separation*
FTP	*File Transfer Protocol*
GB	*Gigabyte*
Gbps	*Gigabits* por segundo
GHz	*Gigahertz*
GT-EWS	Grupo de Trabalho *Early Warning System*
HIDS	*Host Intrusion Detection Systems*

HTTP	*HyperText Transfer Protocol*
HTTPS	*HyperText Transfer Protocol Secure*
I-SID	*Instance Service Identifier*
IA	Inteligência Artificial
IBM	*International Business Machines*
ICMP	*Internet Control Message Protocol*
ICMPv4	ICMP versão 4
ICMPv6	ICMP versão 6
Id	Número de Identificação
IDPS	*Intrusion Detection Prevention Systems*
IDS	*Intrusion Detection Systems*
IETF	*Internet Engineering Task Force*
IoT	Internet *of Things*
IP	*Internet Protocol*
IPS	*Intrusion Prevention Systems*
IPv4	IP versão 4
IPv6	IP versão 6
IRC	*Internet Relay Chat*
ISDS	*Intelligent Security Defensive Software*
ISP	*Internet Service Provider*
ITAC	*Intrusion Tolerance Based on Autonomic Computing*
Kbps	*Kilobits* por segundo
LAN	*Local Area Network*
LAND	*Local Area Network Denial*
LISP	*Location/ID Separation Protocol*
M-AID	*Middleware for Anomaly-based Intrusion Detection*
MAC	*Multi-objective Analysis Controller*
MAPE-K	*Monitor*, *Analyse*, *Plan*, *Execute and Knowledge*
MB	*Megabyte*
Mbps	*Megabits* por segundo
MDAC	*Microsoft Data Access Components*
ML-IDS	*Multi-Level Intrusion Detection System*
MPLS	*Multiprotocol Label Switching*
MS-CHAP	*Microsoft Challenge-Handshake Authentication Protocol*
NAT	*Network Address Translation*
ND	*Neighbor Discovery*

NIDS	*Network Intrusion Detection Systems*
NSA	*National Security Agency*
O:MIB	*Object-oriented Management Information Base*
Of-IDPS	*OpenFlow Intrusion Detection and Prevention System*
ONF	*Open Networking Foundation*
OVAL	*Open Vulnerability Assessment Language*
P2P	*Peer-to-peer*
PBB	*Provider Backbone Bridging*
PDF	*Portable Document Format*
pps	pacotes por segundo
PRODAM	Processamento de Dados Amazonas S/A
QoS	*Quality of Service*
RDS	Redes Definidas por Software
RNP	Rede Nacional de Ensino e Pesquisa
RR	*Resource Records*
SAAF	*Self-Adaptive Authorization Framework*
SCTP	*Stream Control Transmission Protocol*
SMB	*Server Message Block*
SMTP	*Simple Mail Transfer Protocol*
SOHO	*Small Office / Home Office*
SQL	*Structured Query Language*
TCP	*Transmission Control Protocol*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
TI	Tecnologia da Informação
TLS	*Transport Layer Security*
ToS	*Type of Service*
TTL	*Time to Live*
UDP	*User Datagram Protocol*
UFBA	Universidade Federal da Bahia
URL	*Uniform Resource Locator*
UTFPR	Universidade Tecnológica Federal do Paraná
VAS	*Vulnerability Awareness System*
VESPA	*Virtual Environments Self-Protecting Architecture*
VLAN	*Virtual LAN*
VM	*Virtual Machine*
VSK	*Virtual Security Kernel*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Práticas como BYOD (*Bring Your Own Device*) tornam as redes de computadores locais mais dinâmicas e amigáveis para o usuário final, mas em contrapartida, adicionam novas vulnerabilidades que são utilizadas frequentemente contra a segurança das redes de computadores (Meng *et al.*, 2015; Symantec, 2015). O uso de mecanismos de segurança tradicionais, tais como *firewall* e IDS (*Intrusion Detection Systems*), normalmente exigem dedicação e perspicácia do administrador, que precisa constantemente analisar o histórico de uso da rede e alertas de segurança para manter um nível de segurança condizente com as ameaças. Todavia, muitas vezes o responsável pela LAN (*Local Area Network*) acumula cargos e por isto não tem tempo hábil para gerenciar a segurança de maneira apropriada. Em casos mais extremos, algumas LANs não possuem um responsável direto e/ou competente pela segurança do ambiente. Como consequência, a detecção e o tempo de reação contra as ameaças é muitas vezes elevado, o que pode comprometer o bom funcionamento da rede por longos períodos de tempo.

Um agravante a esses ambientes de redes mais heterogêneos e em alguns casos pouco assistidos por seus administradores, é que o número de ciberataques contra organizações têm aumentado a cada ano (Meng *et al.*, 2015; Percoco, 2013; Sophos, 2014a; Trustware, 2016). Em especial, observa-se o crescimento do número de ataques destinados e originados à LANs (Cisco, 2016; Symantec, 2015). Ainda somado a esse cenário, muitas ciberameaças atualmente são complexas e constituídas de várias etapas (Feily *et al.*, 2009; Rodríguez-Gómez *et al.*, 2013; Trustware, 2014, 2016). Portanto, manter a segurança de ambientes computacionais modernos manualmente tem se tornado um processo complexo.

Neste contexto de retirar parte do ônus do administrador em manter a segurança das LANs, esta tese apresenta uma arquitetura que emprega CA (Computação Autonômica) para criar e autogerenciar regras de segurança que têm por finalidade mitigar problemas de segurança em redes de computadores. Tais regras são formadas pelo processamento de históricos de alertas organizados em um esquema similar à memória humana. As regras de segurança criadas são aplicadas à rede por meio de técnicas de RDF (Redes Definidas por Software), que também dão suporte à criação dos alertas de segurança em conjunto com sistemas de segurança legados.

1.1 Considerações preliminares

Há evidências da necessidade de aprimorar e prover mecanismos de segurança voltados às necessidades das redes de computadores, em especial para as de menor porte (Sophos, 2014a; Symantec, 2013; Trustware, 2016). Muitas soluções de segurança são direcionadas para as grandes organizações, que podem inviabilizar a implantação para redes de pequeno porte. Além disso, os mecanismos de segurança atuais podem ser afetados pelo tráfego elevado e heterogêneo das LANs. Destacam-se nesse caso, o tráfego normal de operação das organizações e o conteúdo acessado pelos usuários da rede, como por exemplo, o acesso a conteúdos multimídia e redes sociais. Outra dificuldade para manter-se a segurança de LANs atualmente é o conceito de BYOD, que consiste no uso de dispositivos móveis pessoais para acesso a recursos da organização, o que facilita a penetração de softwares

maliciosos (Anderson, 2012; Disterer e Kleiner, 2013).

O uso de IDS é uma solução para detectar ameaças, entretanto o número de alertas gerados pode ser alto e com grande índice de falsos positivos, o que pode inviabilizar tomadas de decisões precisas. Porém, isso pode ser corrigido caso haja integração com alguma ferramenta que tenha uma visão global da rede. As RDS viabilizam o gerenciamento e acesso a novos fluxos de informação de LANs. Com o OpenFlow (McKeown *et al.*, 2008), por exemplo, é possível que *switches* insolem fluxos de redes, bloqueiem conexões indesejadas, implementem QoS (*Quality of Service*), dentre outros, utilizando para isso informações que vão desde a camada de enlace até a camada de transporte (Tanenbaum e Wetherall, 2013), o que não é possível em *switches* que não utilizam o conceito de RDS. Já a CA permite que sistemas trabalhem sem a necessidade de interação humana, o que é possível através de ciclos autonômicos que monitoram o ambiente, e caso hajam ameaças ao equilíbrio do sistema, medidas são criadas e aplicadas no ambiente por meio de atuadores. Por fim, o sistema autonômico continua monitorando e verificando se as medidas surtiram efeito, em um ciclo infinito (Hariri *et al.*, 2006; Huebscher e McCann, 2008; Kephart e Chess, 2003).

Portanto, através do uso de CA é possível criar um sistema que exige pouca ou nenhuma intervenção humana para a resolução de problemas de segurança em LANs. Tal sistema autonômico, pode fazer uso de métodos de RDS para monitorar o ambiente de rede e para executar ações que objetivem sanar problemas que estejam afetando a rede. O sistema autonômico ainda pode utilizar de diversas fontes de informações disponíveis para detectar os ataques à segurança. Incluindo fontes de dados não exploradas, como por exemplo, mensagens a respeito de cibersegurança postadas em redes sociais. Com isso seria possível fornecer mecanismos de baixo custo para aprimorar a segurança de pequenas organizações e viabilizar o desenvolvimento de novos mecanismos de segurança.

1.2 Objetivos

O objetivo principal desta tese é propor e desenvolver uma arquitetura autonômica que mantenha a segurança de redes de computadores, exigindo o mínimo possível de interação humana.

Então, para alcançar o objetivo principal desta tese, é proposta uma arquitetura autonômica que permita através de sensores observar os estados da rede, bem como possíveis ameaças. Caso a rede apresente desequilíbrios, principalmente ocasionados por ciberataques. A arquitetura gera autonomicamente ações que devem ser aplicadas na rede através de atuadores, para desta forma mitigar as ameaças e normalizar o funcionamento da rede.

Para desenvolver a arquitetura autonômica proposta e colaborar com o estado da arte, a presente tese também apresenta os seguintes objetivos específicos:

- Pesquisa de métodos que permitam extrair informações a respeito da rede local. Tais informações devem contemplar tanto as ciberameaças que afetam a rede quanto dados referentes ao uso da rede. Esse tipo de informação é fundamental para que a arquitetura consiga de forma rápida e assertiva reagir contra ameaças ao equilíbrio da rede;

- Criação de métodos que possibilitem extrair alertas de segurança de fontes externas à rede local. Mais especificamente que permitam identificar alertas de segurança que são postados em mensagens de redes sociais. Esta tese levanta a hipótese que as redes sociais podem ser uma nova e rica fonte de informações a repeito de ciberameaças. Podendo servir, no mínimo, para manter administradores de redes informados a respeito de problemas de segurança que afetam as redes mundo a fora;

- Criação e desenvolvimento de métodos que possibilitem que a arquitetura autonômica proposta analise o histórico de uso da rede e de alertas de segurança. Fundamentada nessa análise, a arquitetura deve autonomicamente gerar/manter regras de segurança que mitiguem ciberataques, previnam contra ameaças futuras e evitem que *hosts* e/ou serviços de rede sejam influenciados negativamente pela própria defesa autonômica.

- Implementação de métodos que possibilitem criar atuadores que alterem o comportamento da rede para mitigar os ciberataques identificados pela arquitetura. Também devem ser pesquisados métodos/contramedidas que permitam reagir adequadamente e proporcionalmente em casos de suspeitas de ataques ou aos diferentes tipos de ataques que a rede pode sofrer.

Desta forma, espera-se criar LANs que tenham a capacidade de se auto-proteger contra as ciberameaças que podem comprometer os muitos recursos que estão disponíveis nas redes de computadores atuais.

1.3 Contribuições

Com o intuito de criar uma arquitetura autonômica que proteja redes de computadores de ciberataques. A presente tese inova, pois propõe a criação e desenvolvimentos de métodos que empregam algoritmos de análise de associação para processar históricos de uso da rede e alertas de segurança, extraindo então regras de segurança que são aplicadas autonomicamente na rede, por intermédio de RDS. Outra contribuição é a criação de um método inspirado na estrutura da memória humana, que permite gerar autonomicamente regras de segurança que:

- Mitiguem problemas de segurança na rede;
- Previnem contra ameaças futuras;
- Evitam que *hosts* e/ou serviços da rede sejam influenciados por ciberataques e pela própria defesa autonômica, em caso de falsos positivos.

A arquitetura autonômica proposta pode utilizar várias fontes de informações como sensores para identificar ciberameaças que estejam comprometendo o bom funcionamento das redes. Sendo que tais informações podem vir de fontes de dados já tradicionais, tal como alertas gerados por IDS. Ou serem obtidas de fontes mais contemporâneas, com por exemplo da tecnologia OpenFlow.

Outra inovação proposta por esta tese é a geração de alertas de segurança a partir de mensagens que relatam ciberameaças e que são postadas na Internet através de redes sociais. Neste sentido, a intensão desta tese é verificar se é possível extrair alertas de segurança de mensagens postadas em redes sociais, para futuramente tentar correlacionar automaticamente tais alertas com as informações da rede local (estatísticas de uso da rede, alertas de IDS, dentre outras), o que poderá auxiliar na elucidação de suspeitas a respeito de ciberataques.

Uma das propostas desta tese é utilizar OpenFlow tanto no papel de sensor quanto de atuador da arquitetura autonômica. Pois, o OpenFlow tem sido utilizado como suporte para novos mecanismos de segurança (SRI e ATM, 2013). Soluções usando a combinação entre OpenFlow e IDS foram propostas (Nagahama *et al.*, 2012; Xing *et al.*, 2013), mas não há uma análise crítica do impacto e limitações na implantação dessas soluções, nem pesquisas que unem essas tecnologias com CA para mitigar problemas de segurança em LANs. Assim, esta tese contribui analisando na prática as vantagens e desvantagens do emprego da tecnologia OpenFlow em um cenário de rede autonômico que também utiliza sistemas de segurança legados, tal como o Snort NIDS (*Network Intrusion Detection Systems*).

Outra contribuição, é que apesar desta tese abordar a detecção de ciberameaças o foco principal da presente tese é na reação autonômica. Isso é importante pois muitos trabalhos existentes, na área de CA aplicada a segurança (*self-protection*), exploram a detecção de problemas, mas poucos investigam como reagir e mitigar tais ameaças (Yuan *et al.*, 2014).

Resumindo, o presente trabalho apresenta como contribuições: (i) o desenvolvimento de uma arquitetura e métodos que permitam integrar CA e RDS para mitigar problemas de segurança em LANs; (ii) a combinação de informações de fontes heterogêneas internas e externas ao ambiente de rede local, para a geração e reação autonômica à incidentes de segurança; (iii) o desenvolvimento de métodos que permitam reagir dinamicamente e proporcionalmente aos diferentes tipos de problemas

de segurança; (iv) um protótipo de aplicação de segurança; (v) e a investigação da efetividade das RDS em aplicações de segurança.

A prova de conceito da arquitetura proposta é dada através da implementação, experimentos e análises de uma ferramenta de segurança denominada Of-IDPS (*OpenFlow Intrusion Detection and Prevention System*), que tem como intuito avaliar a eficiência, limitações e impactos na detecção e reação as ameaças de segurança usando fontes de informação distribuídas, RDS e CA.

1.4 Organização do trabalho

Este documento discute no Capítulo 2 conceitos necessários para o entendimento deste trabalho, no Capítulo 3 os trabalhos relacionados e no Capítulo 4 a metodologia e arquitetura proposta. O Capítulo 5 apresenta os experimentos e seus resultados e o Capítulo 6 apresenta as considerações finais e trabalhos futuros.

Capítulo 2

Conceitos

Nesse capítulo são apresentados os conceitos necessários para o entendimento da proposta.

2.1 Segurança de computadores

A sociedade contemporânea esta cada vez mais dependente dos computadores, que são utilizados no cotidiano das pessoas para agilizar tarefas relacionadas ao trabalho e em momentos de lazer (Tanenbaum, 2007). As tecnologias advindas dos computadores provêm inúmeros benefícios, sendo um dos principais o acesso a Internet, através da qual é possível usufruir de informações globalizadas, bem como desfrutar de incontáveis serviços, tal como, redes sociais. Entretanto, com o advento dos computadores também surgem novos problemas, principalmente relacionados à segurança. Desta forma, a cada dia que passa a população sofre mais com pragas e ameaças digitais, que são disseminadas principalmente através da Internet (Chang *et al.*, 2013) e afetam negativamente o funcionamento dos sistemas informatizados (Bishop, 2004; Stallings e Brown, 2014; Tanenbaum e Wetherall, 2013).

Quando se discute segurança de computadores muitas pessoas lembram de vírus de computadores (Talib *et al.*, 2010), pois esses causam transtornos e afetam diretamente muita gente, mas as ameaças computacionais não ficam restritas a um único problema (vírus), pelo contrário é uma área vasta e complexa, pois envolve aspectos tecnológicos, gerenciais e educacionais (Bishop, 2004; Stallings e Brown, 2014). Dessa forma, a segurança deve ser zelada por todas as especialidades e elementos da computação: hardware, sistemas operacionais, redes de computadores, softwares aplicativos, banco de dados, dentre outros. Como agravante, a segurança está ligada a questões fora do âmbito computacional: cultura, educação, treinamento dos usuários que utilizam os sistemas (Rao e Pati, 2012), desastres naturais e outras circunstâncias que muitas vezes são imprevisíveis e incontroláveis.

Sendo a área de segurança de computadores tão vasta e complexa, os gestores de ambientes computacionais devem ter ciência de que não há como garantir 100% de segurança (Bishop, 2004; Stallings e Brown, 2014), mesmo assim tais gestores precisam utilizar métodos que permitam alcançar um nível aceitável de segurança, já que atualmente muitas pessoas necessitam dos serviços oferecidos por esses ambientes computacionais. A aplicação desses métodos de segurança deve ser bem dosada e direcionada, buscando sempre atingir um bom índice de custo/benefício para tornar a manutenção da segurança factível.

Apesar de existirem muitos pontos a serem cobertos para se manter a segurança de computadores, há um que merece atenção especial, que é a informação (ISO27001, 2013), pois atualmente a informação tem um imenso valor. Em muitos casos a informação é mais importante e valiosa que os próprios recursos de hardware utilizados para tratá-la (Cisco, 2014). Em resumo existem três itens básicos a serem mantidos quanto à segurança da informação (Bishop, 2004; Stallings e Brown, 2014), que são:

- Confidencialidade: Significa que algumas informações são secretas e que somente quem for autorizado deve ter acesso à informação. Todavia, não são todas informações que requerem o

item confidencialidade, pois alguns dados são públicos e qualquer um pode ter acesso. Para as informações que devem ser mantidas em sigilo é possível utilizar meios tecnológicos tais como, controle de acesso (*login*/senha) e criptografia;

- Disponibilidade: Não basta ter a informação. A mesma deve estar disponível quando for requerida, ou seja, disponibilidade é garantir que a informação ou até mesmos recursos vão estar prontos para uso quando forem solicitados. É possível tornar os sistemas computacionais mais disponíveis através da redundância de recursos, em contrapartida a replicação de informação/recursos pode encarecer a manutenção do sistema. Um exemplo simples e eficiente de técnica de disponibilidade são as cópias de segurança (*backups*).

- Integridade: Esse item representa proteger as informações contra alterações não autorizadas ou erros que possam comprometer os dados processados, armazenados ou em trânsito. Para se manter a integridade dos dados são necessários mecanismos que garantam que a informação acessada é correta e verdadeira, isso por exemplo é possível utilizando técnicas computacionais, como algoritmos *hash*. Segundo Bishop (2004), o item integridade também incorpora a tarefa de verificar se a origem dos dados é verdadeira e se quem está acessando a informação também não é um impostor, essa tarefa pode ser possível empregando-se certificados digitais ou técnicas de biometria.

É importante observar que a manutenção da segurança é tão meticulosa que um item da informação pode comprometer outro item. Por exemplo, não é incomum que administradores de TI (Tecnologia da Informação) decidam empregar um esquema de senhas fortes em seus equipamentos e depois esqueçam tal senha, o que consequentemente torna o acesso ao sistema indisponível enquanto a senha não for lembrada/recuperada, isto é, aplicando-se um alto grau de confidencialidade em um sistema é plausível afetar a disponibilidade do mesmo.

A segurança da informação pode ser complementada com técnicas como autenticidade, não-repúdio e auditoria (Bishop, 2004; Stallings e Brown, 2014). A autenticidade representa o emprego de métodos que impeçam que pessoas não autorizadas acessem os recursos computacionais. Não-repúdio é a utilização de técnicas que impeçam que pessoas neguem ou contestem ações garantindo assim a autoria de ações em ambientes digitais. A auditoria é o processo de analisar, avaliar e identificar eventos e ações que tenham ocorrido no sistema, bem como seus respectivos autores.

Nas seções seguintes serão apresentadas as principais ameaças a segurança da informação, contra-medidas aos problemas de segurança e por fim alguns desafios na área de segurança de computadores.

2.1.1 Ameaças a segurança da informação

Uma ameaça à segurança da informação simboliza algum tipo de ação que pode comprometer alguns dos itens básicos da informação (confidencialidade, disponibilidade e integridade). De forma mais abrangente, ameaça representa alguma violação em potencial à segurança, ou seja, algo que possa comprometer o bom funcionamento do sistema. Normalmente a ameaça está ligada a algum tipo de vulnerabilidade, que é um ponto fraco do sistema e que pode ser atacado. Um atacante é quem usa das vulnerabilidades para efetivar as ameaças e concretizar os ataques contra a segurança.

Segundo Bishop (2004) existem basicamente quatro tipos de ameaças em ambientes computacionais, que são: (1) divulgação ou acesso não autorizado de informações; (2) fraude; (3) perturbação ou interrupção de serviços; e (4) controle não autorizado do sistema. Derivadas dessas surgem ainda ameaças ou ataques compostos que podem afetar a segurança de computadores, como por exemplo:

- Um atacante pode capturar informações que são transmitidas por meio de redes de computadores, com o objetivo de bisbilhotar (*snooping*) os dados em busca de informações importantes, como senhas por exemplo. Esse tipo de ação caracteriza uma ameaça de divulgação ou acesso não autorizado de informações;

- O ataque anterior ainda pode ser incrementado, de forma que além de interceptar e capturar os pacotes de rede, o atacante ainda altere e envie tais dados modificados para a vítima. O que representa uma fraude, pois a vítima não receberá os dados originais, mas sim os dados manipulados pelo atacante. Esse tipo de ataque é normalmente conhecido como homem no meio do caminho (*man-in-the-middle*);

- Um ataque relativamente comum é quando o atacante tenta enganar a vítima fazendo-se passar por outra entidade (*spoofing*). Nesse uma pessoa mal intencionada, por exemplo, pode se passar por uma agência financeira e ficar esperando que os clientes se conectem no servidor falso, para assim roubar informações bancárias. Também, é possível mascarar (*masquerading*) endereços alterando origem e/ou destino para confundir ou camuflar ações maliciosas. Para tanto pode-se, por exemplo, utilizar um *proxy* anônimo para dificultar a identificação do atacante;

- Alguns ataques têm por objetivo causar atrasos (*delay*) nos serviços providos pelos computadores, como por exemplo na entrega de informações. Ataques desse tipo normalmente tentam perturbar o funcionamento do sistema ou até mesmo interrompê-los. Atualmente é comum o uso de técnicas que ameaçam a disponibilidade de recurso, tais ataques são conhecidos como negação de serviço e têm por objetivo tornar indisponíveis recursos computacionais;

- Outras duas ameaças presentes principalmente em transações eletrônicas são os atos de repudiar a origem (*repudiation of origin*) ou de negar o recebimento de ações ou informações (*denial of receipt*). Um exemplo de repudiar a origem é o caso de uma pessoa mal intencionada realizar alguma transação eletrônica, tal como, uma compra pela Internet e depois de já estar usufruindo da compra, tentar trapacear dizendo que não foi o próprio quem fez a compra eletrônica, portanto, o atacante tenta obter benefícios, tal como o ressarcimento do valor da compra, dizendo falsamente que foi "roubado". Usando a mesma exemplificação, o ato de repudiar o recebimento poderia ser o atacante receber o produto e alegar falsamente que não o recebeu, exigindo assim a entrega do produto novamente - esse tipo de problema pode surgir em vendas totalmente eletrônicas, no qual o produto (jogo, livro, etc) também é digital e o comprador tem, por exemplo, o direito de fazer um número limitado de *downloads* do produto e ele excede esse número, mas alega que não. Métodos de não repúdio tentam inibir esse tipo de ameaça.

As ameaças e ataques apresentados anteriormente são representações reais, porém um pouco abstratas dos problemas que podem afetar os ambientes computacionais. A seguir serão descritos alguns problemas mais práticos que normalmente prejudicam tais ambientes.

2.1.1.1 Programas maliciosos

Existem vários programas maliciosos que são desenvolvidos por pessoas mal intencionadas visando afetar a segurança dos ambientes computacionais (Bishop, 2004). Tais programas são conhecidos como *malwares*, que é a contração para *malicious software* ou em português softwares maliciosos. Os *malwares* causam muitos transtornos em ambientes informatizados, pois comprometem serviços e afetam o acesso às informações. Atualmente há várias pesquisas na área acadêmica e muito esforço da indústria para combater esse mal, mesmo assim soluções conhecidas, como antivírus, IDS, *firewalls*, dentre outros, ainda apresentam limitações na mitigação dos sintomas causados pelos *malwares* (Chang *et al.*, 2013; Marpaung *et al.*, 2012). Os cavalos de Tróia, vírus e *worms* são os tipos mais comuns de *malwares* (Karresand, 2003).

O cavalo de Tróia (*Trojan horse*) é um programa malicioso que finge ser um programa idôneo (Bishop, 2004; Karresand, 2003). Quando o usuário executa o programa, que tem embutido o código malicioso, uma ação inesperada é disparada. Os cavalos de Tróia normalmente são utilizados em ações como: disponibilizar serviços de rede que permitem que atacantes acessem ou controlem suas

vítimas (*backdoor*); capturar e enviar para o atacante ações da vítima, tal como teclas digitadas (*keylogger*) ou cliques do mouse (*mouselogger*); dentre outras inúmeras atividades maliciosas.

O tipo de *malware* mais conhecido é o vírus (Talib *et al.*, 2010), que nada mais é do que um programa composto normalmente de instruções maliciosas (Sanok, 2005). O vírus de computador e o cavalo de Tróia são semelhantes, mas com a diferença de que o vírus tem como característica principal a reprodução ou replicação do seu código malicioso, coisa que o cavalo de Tróia não faz. Conforme Karresand (2003), os vírus não dependem da vítima para iniciar sua execução, o vírus pode fazer uso de alguma vulnerabilidade dos sistemas/aplicações para infectar a máquina. Assim, quando um vírus infecta um computador esse tenta fazer cópias de si mesmo em outros arquivos, para posteriormente efetivar ataques contra a segurança do sistema. Hoje em dia existe uma infinidade de vírus de computadores (Bishop, 2004), que podem infectar várias partes de um sistema computacional, tal como: o setor de inicialização (*boot*) do computador, arquivos, a memória principal, dentre outros. Atualmente os vírus de computadores se tornaram tão eficazes que utilizam-se de técnicas para dificultar a sua detecção, algumas dessas técnicas são: interceptar e modificar chamadas do sistema operacional para ocultar o vírus (*stealth*); alterar o código do vírus utilizando técnicas de criptografia; alterar parte do código de programação do vírus por um código diferente mas equivalente, o que é conhecido como vírus polimórfico. Também, existem vírus de macro, que são executados por interpretadores (não compiladores) (Tanenbaum, 2007), assim esse tipo de vírus pode ser multiplataforma e afetar diferentes sistemas operacionais.

O conceito de *worm* é semelhante ao de vírus, com a diferença de que o vírus depende de alguma aplicação secundária para conseguir se replicar, em contra partida o *worm* faz sua replicação de maneira independente (Sanok, 2005). Portanto, o vírus precisa de um hospedeiro para se proliferar, já o *worm* é auto-suficiente e consegue se reproduzir sozinho. Outra grande diferença é que o *worm* consegue infectar outros computadores via rede, enquanto o vírus fica restrito ao *host* (Karresand, 2003). Conforme Bishop (2004), o vírus só pode infectar outros programas, já os *worms* conseguem se propagar por vários computadores.

Atualmente um dos tipos de *malware* mais utilizados e perigosos são os que dão suporte às *botnets* (Feily *et al.*, 2009; Rodríguez-Gómez *et al.*, 2013; Sanok, 2005). As *botnets* são computadores com a segurança comprometida, ligados via rede e que servem de base para uma grande variedade de ataques à segurança de diversos sistemas. Uma *botnet* é uma rede composta por computadores chamados de *bots* que são controlados remotamente por um atacante intitulado *botmaster*. Os *bots*, também são conhecidos como zumbis[1], são responsáveis por executar ações maliciosas determinadas pelo *botmaster*, daí o nome *bots* que é derivado de *robot*[2], pois esses são robôs prontos para executar ações automáticas a mando de seu mestre. O *botmaster* utiliza um canal de comunicação para controlar os *bots*, e é justamente a forma de enviar comandos e controlar os *hosts* comprometidos que normalmente diferencia uma *botnet* de outra. Os métodos ou canais de comunicação normalmente utilizados são: IRC (*Internet Relay Chat*), HTTP (*HyperText Transfer Protocol*), DNS (*Domain Name System*) ou P2P (*Peer-to-peer*) (Feily *et al.*, 2009). A relação das *botnets* com *malwares* é que os computadores que fazem parte das *botnets* são infectados por algum software malicioso. Um dos objetivos desse software é infectar o maior número de computadores possível, ao redor do mundo, pois isso ajudará nos ataques e dificultará contra-medidas para detê-los.

Um ponto interessante dos *malwares* de *botnets* é que esses normalmente possuem mecanismos que permitem atualizar e configurar o software malicioso que dá suporte à *botnet*. Essa atualização é muito importante na manutenção da *botnet*, pois permite que o atacante instale versões mais atuais do *malware* e assim inclua novas funcionalidades ou técnicas para ocultar o *malware*. Também, através desse canal de atualização, é possível alterar configurações dos *bots*, como: o endereço IP do servidor que controla os *bots*, ou servidor DNS. Dessa forma o *botmaster* pode facilmente mudar a sua localização (trocar de servidor), caso as autoridades descubram o servidor que deu origem aos ataques.

As *botnets* servem de exemplo para ilustrar o quão complexos são os ataques de segurança atual-

[1] No contexto das crenças vodu (*voodoo*)

[2] Robô em português

mente, pois as *botnets* empregam vários tipos de ameaças para conseguir comprometer a segurança de ambientes computacionais (Rodríguez-Gómez *et al.*, 2013). Seguindo o ciclo de vida de uma *botnet*: primeiramente são utilizados *malwares*, para infectar computadores; depois tais computadores ficam sob comando do *botmaster*, que pode dentre outras coisas, acessar indevidamente informações das máquinas infectadas ou instalar/atualizar outros programas maliciosos; Por fim, tais computadores são utilizados em um ataque maior, tal como negação de serviço e envio de propagandas pela Internet (*spams*) (Feily *et al.*, 2009). Contudo é importante notar que as *botnets* além de afetar o alvo do ataque maior, também afetam negativamente os *hosts* e redes onde estão os *bots*, pois durante os ataques são consumidos indevidamente recursos como processador, memória e largura de banda da rede.

2.1.1.2 Negação de serviços

Como já mencionado anteriormente uma das principais ameaças aos sistemas computacionais é o de atraso ou negação de serviços, que tem por finalidade lesar um dos itens básicos da segurança, que é a disponibilidade. De acordo com pesquisas (Kumar e Selvakumar, 2009; Zargar *et al.*, 2013) esse tipo de ataque tem crescido muito nos últimos anos. Na grande maioria das vezes tais ataques são representados por duas siglas: DoS e DDoS.

Um ataque DoS tem por objetivo tornar indisponíveis, para usuários legítimos, os serviços/recursos de computadores ou sistemas (Peng *et al.*, 2007). Tais recursos podem ser de vários contextos, como, hardware, sistema operacional, aplicativo ou rede. Portanto, um ataque DoS irá tentar exigir em demasia partes do computador como processador, memória, placa de rede, dentre outras, com o intuito de prejudicar os usuários do sistema. Por exemplo, em um ataque DoS ao processador, o atacante irá tentar esgotar os recursos do processador executando processos maliciosos, de forma que processos de usuários legítimos demorem muito para serem executados ou até mesmo não tenham chance de serem processados. O ataque DoS ao processador pode se dar por meio de um *malware*, que tem por função executar algum cálculo matemático que exija muito do processador, só para deixá-lo muito ocupado e assim atrapalhar o processamento de outros aplicativos. Em um ataque DoS de rede, um atacante pode enviar milhares de pacotes de rede para a vítima. O intuito é que o computador da vítima, ao tentar tratar os pacotes maliciosos utilize tantos recursos que venha a apresentar problemas de desempenho na rede.

Os ataques de DoS podem ser classificados de forma bem abrangente como: ataques de esgotamento de recursos e de esgotamento de largura de banda da rede (Kumar e Selvakumar, 2009; Peng *et al.*, 2007). Um ataque de esgotamento de recursos (*resource deplation*), tem por objetivo estressar e tornar indisponíveis os recursos da vítima, tal como processador e memória. Já o ataque de esgotamento da largura de banda (*bandwidth depletion*) tem por finalidade saturar os recursos de rede da vítima, isto é, atacar os enlaces de comunicação que a vítima utiliza. Esse ataque pode ser mais devastador que o de esgotamento de recursos, pois como efeito colateral o ataque de esgotamento de largura de banda pode comprometer, além da vítima alvo, equipamentos e fluxos de redes de outros *hosts* que também utilizam a mesma infraestrutura, logo esse ataque faz, mesmo que indiretamente, mais vítimas.

O DDoS é uma especialização do DoS. A diferença é que em ataques DDoS são utilizadas múltiplas fontes para executar o ataque à vítima (Peng *et al.*, 2007; Zargar *et al.*, 2013). Ou seja, quando o ataque ocorre partindo de uma única origem é intitulado DoS, já quando o ataque parte de várias fontes, de preferência simultaneamente, o ataque passa a levar o nome de DDoS. No exemplo do ataque de rede DoS, apresentado anteriormente, a fonte do ataque erá um único *host*, o que caracteriza um ataque de negação de serviço simples. Para transformar esse mesmo ataque em um DDoS basta aumentar o número de *hosts* atacantes. A grande vantagem do DDoS, para o atacante, é que o efeito do ataque é potencializado pelo número de máquinas que estão colaborando no ataque, o que possivelmente irá ajudar a atingir o objetivo do ataque mais rapidamente. Normalmente ataques DDoS estão intimamente relacionados com redes de computadores, pois esses ambientes são propícios a ataques distribuídos. Como o ataque DDoS é uma variação do DoS é comum chamá-lo simplesmente de DoS - o mesmo será feito algumas vezes nesse trabalho - mas a maior parte dos

ataques hoje em dia são do tipo DDoS (Kumar e Selvakumar, 2009; Zargar *et al.*, 2013). Segundo Peng *et al.* (2007) os tipos de ataques DDoS mais conhecidos são os de largura de banda baseados em protocolo e de largura de banda baseados em aplicação.

Ataques de largura de banda baseados em protocolo (*protocol-based bandwidth attacks*), são ataques que fazem uso de vulnerabilidades presentes em protocolos de rede e podem, por exemplo, serem utilizados em ataques DoS com uma única fonte de ataque ou com um número reduzido de atacantes. Uma exemplificação clássica desse tipo de ataque é o de inundação SYN (SYN *flood*), que faz uso de peculiaridades do protocolo TCP (*Transmission Control Protocol*) para comprometer recursos da vítima. O ataque SYN se dá com o atacante iniciando uma conexão TCP com o servidor da vítima, mas sem completar apropriadamente a abertura de conexão, o que consome recursos do servidor da vítima (Tanenbaum e Wetherall, 2013). Um dos motivos de não completar a conexão TCP é que os atacantes podem utilizar IPs falsos (*spoof*), o que impossibilita a comunicação entre cliente/servidor, mas atualmente uma variação desse ataque é utilizar as *botnets* para completar a sequência de abertura da conexão TCP, o que consome ainda mais recursos do servidor da vítima. Outro exemplo desse tipo de ataque é o Smurf que utiliza inundação ICMP (*Internet Control Message Protocol*) (ICMP flood). Durante um ataque Smurf, o atacante envia pacotes ICMP do tipo *echo request* destinados a endereços *broadcast* da Internet, o grande truque é que o atacante máscara o endereço IP de origem pelo endereço da vítima, ou seja, o endereço de origem não é o do atacante, mas sim da vítima. A ideia do ataque é que o atacante envia um único pacote destinado a várias máquinas por meio do endereço IP de *broadcast*, as máquinas que receberem tal pacote irão responder naturalmente com um pacote ICMP do tipo *echo reply*, só que responderão para a vítima, já que o atacante forja o pacote com o endereço da vítima. Essa técnica utilizada pelo Smurf é conhecida como amplificação e reflexão, já que um único pacote enviado pelo atacante é amplificado (multiplicado) pelos *hosts* que tratarem tal pacote, por fim os pacotes de resposta são todos refletidos (transmitidos) para a vítima, comprometendo recursos dessa e da rede local.

Ataques de largura de banda baseados em aplicação (*application-based bandwidth attacks*) são ataques que exploram peculiaridades de aplicações para ameaçar a disponibilidade de serviços. Para isso, o atacante normalmente força a aplicação da vítima a realizar várias operações que exijam muito do ambiente computacional, de forma a sobrecarregá-lo, fazendo com que seja comprometido o processamento de pedidos de clientes legítimos. Um exemplo desse tipo de ataque é o de inundação HTTP (HTTP *flood*). Esse ataque consome os recursos da máquina porque o HTTP utiliza o protocolo de transporte TCP, o que já configura um ataque de inundação SYN. Só que o ataque HTTP exige ainda mais do servidor, pois agora esse precisa tratar as exigências do TCP e enviar informações do protocolo HTTP, tal como o envio de páginas Web, o que certamente gasta mais processador, memória e largura de banda do servidor. Só que o ataque de inundação HTTP pode ser potencializado se for configurado para requisitar determinados tipos de conteúdo dos servidores HTTP.

Nesse contexto é possível variar a granularidade dos conteúdos requeridos durante o ataque, ou seja, é possível configurar o ataque para fazer o *download* de grandes arquivos ou requisitar pequenos conteúdos mas em grandes quantidades, tal como fazer o *download* de todo o conteúdo existente em um servidor HTTP, em ambos os casos seria necessário muito esforço da vítima para atender os pedidos dos atacantes. Uma exemplificação de ataque de aplicação que utiliza reflexão e amplificação, é o ataque de amplificação DNS (DNS *amplification attack*). Nesse ataque, o atacante faz uso de *botnets* para fazer pedidos falsos de resolução de nome em servidores *DNS* (Tanenbaum e Wetherall, 2013), só que o atacante altera o pacote de rede para que o endereço IP de origem do pedido DNS seja o da vítima e não o do atacante (*spoof*), tal como no ataque Smurf. Então, a resposta será refletida do servidor DNS para a vítima e também será amplificada, porque é necessário um pacote de tamanho x para o pedido DNS, mas a resposta do servidor é geralmente maior que x, assim pedidos DNS de poucos *bytes* irão gerar respostas de vários *bytes*, o que amplifica o ataque à vítima. O ataque pode ter uma amplificação ainda maior se o atacante conseguir invadir um servidor DNS e criar um registro TXT RR (*Resource Records*)[3] grande. Nesse caso a resolução

[3]Permite inserir textos em registros DNS

será feita sobre o registro TXT RR inserido pelo atacante, assim conforme o exemplo citado por Peng *et al.* (2007) pedidos DNS de 140Mbps (*Megabits* por segundo) gerados pelos *bots* da *botnet* poderão resultar em um fluxo de 10Gbps (*Gigabits* por segundo) para a vítima.

Então, de forma geral em redes de computadores é possível empregar dois modelos de ataques DDoS: O (1) primeiro consiste em enviar pacotes mal formados e/ou que exploram alguma falha ou vulnerabilidade do sistema; o (2) segundo método, tenta enviar fluxos de pacotes "comuns", mas em quantidades extraordinárias, para tentar esgotar os recursos do sistema. Cada método tem suas vantagens e desvantagens, o primeiro tem como vantagem não necessitar de um grande número de máquinas para realizar o ataque, pois explora vulnerabilidades que normalmente permitem consumir naturalmente os recursos das máquinas, em contrapartida esse tipo de ataque pode ser remediado com atualizações de sistemas operacionais ou softwares. O segundo método não faz uso de vulnerabilidades, por isso pode precisar de muitos computadores para conseguir sobrecarregar a vítima, todavia é um tipo de ataque muito difícil de ser mitigado, pois a princípio os pacotes dos atacantes são pacotes comuns, o que torna muito complicado distinguir o atacante do usuário legítimo. Devido a essas características o segundo método é bem interessante para os ataques atuais (Peng *et al.*, 2007) principalmente se forem reforçados pelo uso de *botnets* (Feily *et al.*, 2009; Rodríguez-Gómez *et al.*, 2013) e pelas técnicas de reflexão e amplificação, tal como citado nos exemplos de ataques Smurf e DNS amplificado, tais técnicas ajudam a potencializar o ataque (amplificação) e ainda confunde a vítima, pois na verdade quem está enviando os pacotes do ataque não é o atacante diretamente, mas sim máquinas comuns que estão sendo ludibriadas pelo atacante a enviar pacotes para a vítima (reflexão), o que certamente dificulta muito a mitigação desse tipo de ataque.

Por fim, os problemas de disponibilidade não ocorrem sempre devido a ataques maliciosos, em alguns casos tal problema é ocasionado de forma não proposital. Por exemplo, é natural que uma rede ou servidor seja planejado para uma determinada carga de serviço, provavelmente fundamentada na rotina da rede/servidor, contudo pode ocorrer dessa carga aumentar inesperadamente, devido ao aumento natural do uso dos recursos ou um evento imprevisto, o que resultará em sobrecarga dos recursos da rede/servidor e por consequência indisponibilidade ou atraso de serviços para os usuários desse sistema. Exemplos mais práticos podem ser a rede de uma universidade que é sobrecarregada durante o período de matrícula dos alunos ou os sistemas da Receita Federal que sofrem com a enxurrada de informações que são enviadas nos dias que precedem o prazo final à entrega do imposto de renda. Todavia, a indisponibilidade de serviços é um problema de segurança e deve ser tratada, mesmo se essa ocorrer de forma inconsciente e não proposital.

2.1.1.3 Outros problemas comuns à segurança de computadores

Os *malwares* e os ataques de negação de serviços, apresentados nas seções anteriores, são apenas dois tipos de problemas de segurança. Uma das maiores ameaças a qualquer ambiente computacional é o ser humano, pois normalmente os sistemas computacionais são feitos para servirem as pessoas. Segundo Bishop (2004) um ambiente computacional é formado por hardware, software e pessoas. Assim, a segurança depende incondicionalmente dos seres humanos que desenvolveram, mantém e utilizam os sistemas.

Quando se fala de pessoas que podem comprometer à segurança de computadores é comum lembrar dos *hackers*. Um *hacker* é uma pessoa que possui muito conhecimento a respeito de computadores e por isso consegue utilizar os sistemas computacionais em benefício próprio. Portanto, um *hacker* tem a capacidade de comprometer a segurança de computadores, mas por filosofia, ele não faz isso para o mal, se o fizer esse passa a ser denominado *cracker*, todavia é muito comum usar o termo *hacker* sem fazer essa distinção.

Portanto, o *cracker* representa uma ameaça real a qualquer sistema computacional, mas esses normalmente só invadem ou atacam sistemas se forem motivados, principalmente de forma financeira. Desta forma, geralmente *crackers* não estão interessados em atacar diretamente pequenas e médias empresas, nem pessoas comuns. Mas essas podem ser alvos indiretos dos ataques dos *crackers*. Por exemplo, sendo infectadas por vírus, que darão suporte aos ataques dos *crackers*, através

de *botnets* por exemplo.

Assim sendo, na maioria das vezes a segurança dos sistemas não será afetada por *hackers* e *crackers* genuínos, mas sim por indivíduos (não-*hackers*) que simplesmente fazem uso indiscriminado de ferramentas criadas por *hackers* e *crackers*. De qualquer forma os não-*hackers* representam grande risco à segurança.

Ao contrário do que muitos acreditam, na maioria das vezes o maior vilão contra a segurança de ambientes computacionais não é o *hacker*, mas sim o próprio usuário do sistema, já que grande parte dos problemas de segurança são causados por quem usa o sistema diariamente (Bishop, 2004; Talib *et al.*, 2010) e isso pode ocorrer de forma proposital ou inconsciente.

Os ataques intencionais, que partem dos próprios usuários do sistema, são normalmente chamados de ataques internos e conhecidos por comprometerem seriamente o sistema. Tais ataques são devastadores porque grande parte das medidas de segurança são desenvolvidas para combater ameaças externas e não internas, além do que, tais usuários possuem naturalmente privilégios que podem ser utilizados durante o ataque (Cisco, 2014; Symantec, 2014). Funcionários insatisfeitos são um bom exemplo de usuário interno que pode arquitetar um ataque contra a segurança do sistema. Também existem ameaças à segurança que são geradas de forma inconsciente, pois é natural que o ser humano cometa erros. Da mesma forma, não é incomum que pessoas utilizem o sistema de forma inapropriada, fazendo mal uso dos recursos computacionais, desabilitando ou burlando sistemas de segurança em benefício próprio, ou seja, configurando, gerenciando e utilizando os sistemas de maneira incorreta. Não adianta aumentar drasticamente o nível de segurança de um ambiente informatizado, se os usuários o utilizarem de forma errada. Por exemplo, segundo Trustware (2014) muitas pessoas usam senhas fáceis de serem adivinhadas ou descobertas, tal como: "123456", "qwer1234" ou "password", outra exemplificação ainda mais crítica, citado pela Trustware (2014), é que certos administradores não reconfiguram senhas de equipamentos/sistemas e utilizam usuários e senhas pré-configuradas pelo fabricante (usuário/senha padrão), tal como usuário "admin" com a senha "admin". Portanto, para se conseguir um nível de segurança aceitável é necessário muita atenção a respeito dos usuários que utilizam ou possam utilizar o sistema.

Sabendo que as pessoas podem ser o elo mais fraco da segurança, muitos *crackers* fazem uso de uma técnica chamada engenharia social para lançar seus ataques (Chang *et al.*, 2013). A engenharia social é a arte de enganar, ou seja, o atacante vai ludibriar a vítima de forma que essa execute alguma ação que comprometa sua segurança fornecendo vantagens ao atacante. Um modelo de engenharia social utilizado hoje em dia é o *phishing* (Kaspersky, 2013; Rodríguez-Gómez *et al.*, 2013), que pode utilizar por exemplo, de *e-mails* que falsamente informam o leitor sobre assuntos diversos, tal como: prêmios ganhos, contas/multas a pagar, pendências judiciais, fotos pessoais publicadas na Internet, atualizações de sistemas, dentre uma infinidade de estórias que normalmente induzem a vítima a acessar *links* que provavelmente instalarão algum tipo de *malware*, dando assim início ao ataque planejado.

Como já foi mencionado anteriormente, não existem sistemas 100% seguros, isso porque praticamente tudo o que envolve um ambiente computacional (hardware, software, sistema operacional, protocolos de rede) pode apresentar algum tipo de falha ou defeito. Segundo Bishop (2004), um sistema seguro inicia seu funcionamento em um estado seguro e durante o seu funcionamento o mesmo só altera para estados da mesma natureza. Uma falha de segurança ocorre quando um sistema sai do estado seguro e passa para estados como: inseguro, não autorizado ou desconhecido.

As falhas de segurança podem ser conhecidas ou desconhecidas. As conhecidas geralmente são falhas que tanto o atacante quanto o responsável pelo sistema (desenvolvedor, administrador e usuário) têm ciência. Muitas dessas falhas conhecidas são tão exploradas por atacantes que são chamadas de falhas clássicas, isso ocorre porque mesmo depois da falha ser descoberta demora-se algum tempo para o fabricante corrigi-la adequadamente. Depois disso, ainda é necessário que o administrador/usuário aplique as devidas correções para sanar o problema, o que em alguns casos nem chega a ocorrer (Trustware, 2014).

Além das falhas de segurança conhecidas, existem também falhas que não são conhecidas por ninguém ou por alguma das partes envolvidas no sistema (fabricante, atacante, usuário). Enquanto

o fabricante não corrige efetivamente uma falha, a mesma é conhecida como falha dia-zero ou vulnerabilidade dia-zero (*zero-day vulnerability*) (Chang *et al.*, 2013; Marpaung *et al.*, 2012), que são comumente exploradas por *hackers* e *crackers*, que desenvolvem *malwares* especializados, denominados *exploits zero-day* (Chang *et al.*, 2013; Feily *et al.*, 2009) - também existem *exploits* para falhas clássicas. Vulnerabilidades dia-zero são extremamente perigosas, pois normalmente passam despercebidas pela maioria dos sistemas de segurança, já que esses não têm ciência dessas ameaças e por isso é complicado detê-las.

Atualmente as redes de computadores representam um grande risco a segurança de sistemas computacionais. Hoje basicamente todo computador está ligado a algum tipo de rede, principalmente a Internet, pois essas trazem muitos benefícios, mas em contrapartida são uma grande ameaça contra a segurança (Cisco, 2014). Isso ocorre justamente porque as redes de computadores ligam pessoas, mas como visto anteriormente, quanto mais pessoas tiverem acesso a um sistema maior é a chance de vulnerabilidades de segurança serem exploradas. Ou seja, computadores expostos na Internet podem: (i) ter seus dados roubados por *hackers* e *crackers*; (ii) ser vítimas de *malwares*; (iii) ser fonte[4] ou alvo de ataques de negação de serviço (DoS); (iv) os recursos da rede podem ser utilizados de forma errada (acesso a conteúdos impróprios), trazendo assim prejuízos aos usuários da rede, tal como lentidão (Sophos, 2014a; Trustware, 2014); (v) dentre outros problemas. Dessa forma, as redes são uma porta aberta para vários problemas de segurança, todavia esses problemas devem ser sobrepujados, já que a sociedade moderna está cada vez mais dependente das redes de computadores.

Existem outros problemas e ameaças à segurança da informação ou de computadores além dos relatados aqui (Bishop, 2004; Stallings e Brown, 2014; Trustware, 2014). A seguir serão apresentadas medidas de segurança que ajudam a conter os tais problemas, bem como alguns desafios da área.

2.1.2 Políticas e mecanismos para manter a segurança da informação

Devido ao fato da segurança de computadores ser uma área vasta e depender de aspectos tecnológicos e sociais, é interessante determinar metas a respeito do que vai ser protegido e como isso será feito (Bishop, 2004; Stallings e Brown, 2014). As metas e objetivos de segurança podem ser representados pela política de segurança, que por sua vez pode ser efetivada ou auxiliada por mecanismos de segurança.

A política de segurança é um conjunto de normas que especificam o que pode ou não ser feito. A política também pode ser representada matematicamente utilizando-se uma lista de estados ditos seguros ou inseguros (Bishop, 2004). No âmbito computacional, o termo política de segurança pode ser utilizado em mais de um contexto, mas sempre com um único objetivo: manter um bom nível de segurança por meio de regras que ditam o que é permitido ou proibido fazer com os recursos computacionais e suas informações. Dois contextos comuns da política de segurança são gerenciais e organizacionais, sendo que nesses a política de segurança deve ajudar a determinar o que deve ser protegido, contra quem proteger, quais comportamentos são aceitáveis e quais não são. Isso ajuda a determinar como a equipe de TI e todos envolvidos no processo devem agir para atender aos objetivos e estratégias organizacionais. A política de segurança no contexto organizacional na maioria das vezes é um documento que determina como os recursos computacionais devem ser utilizados (Stallings e Brown, 2014). A política de segurança também pode ser vista em algum contexto mais específico, tal como, regras que determinam quais tipos de pacotes ou serviços de rede podem ou não adentrar a uma rede local, mas o objetivo ainda é proteger os recursos e informações. As políticas de segurança algumas vezes podem ser conflitantes e imprecisas (Bishop , 2004) o que pode dificultar a intercomunicação entre diferentes sistemas de informação. Mesmo assim é recomendável determinar direitos e deveres de cada ambiente computacional para manter o mínimo de segurança.

Mecanismos de segurança são métodos, procedimentos ou ferramentas utilizados para dar suporte às políticas de segurança, ajudando a preservar a confiança e a estabilidade dos ambientes

[4] Tornando-se zumbis de *botnets*.

computacionais (Bishop, 2004). No âmbito computacional é comum o emprego de softwares para se manter a segurança, contudo nem sempre os mecanismos de segurança utilizados serão somente tecnológicos, por exemplo, não adianta utilizar softwares de segurança, se um servidor ficar fisicamente exposto, neste caso é preciso confinar o servidor em uma sala apropriada e bloquear o acesso físico com o uso de uma simples porta com fechadura, da qual somente pessoas autorizadas possuem a chave.

De maneira geral os mecanismos de segurança podem ser representados por três tipos de medidas de segurança ou de combinações dessas, que são (Bishop, 2004; Stallings e Brown, 2014):

- Preventiva: Tem por objetivo evitar que ataques a segurança aconteçam. Mecanismos de segurança preventivos tentam evitar ou inibir que vulnerabilidades ou falhas de segurança sejam exploradas por atacantes;

- Detectiva: Deve perceber o problema de segurança enquanto esse estiver ocorrendo. Mecanismos detectivos podem apenas gerar alertas, para que os responsáveis tomem providências a respeito do problema ou o próprio mecanismo pode reagir automaticamente tentando resolvê-lo (*self-healing*);

- Corretiva: Tem por função manter o sistema em funcionamento, mesmo que problemas tenham afetado a segurança do mesmo. Geralmente são aplicadas medidas corretivas quando mecanismos preventivos e detectivos falharam e o sistema teve sua segurança comprometida.

Existem várias técnicas utilizadas para manter a segurança de ambientes computacionais, bem como ferramentas (hardware e/ou software) que implementam os mecanismos de segurança. Assim, os principais mecanismos e técnicas de segurança são apresentados a seguir:

- Autenticação: É o método utilizado para verificar se algo é verdadeiro. Muito usado para identificar partes de uma transação eletrônica. Para esse fim normalmente são utilizadas técnicas como: usuário/senha, desafio/resposta, biometria, cartões inteligentes (*smart cards*), dentre outros;

- Criptografia: É a arte de tornar informações incompreensíveis, de forma que somente pessoas autorizadas consigam utilizá-las, então é um tipo de confidencialidade. Segundo Bishop (2004) a palavra criptografia significa "*escrita secreta*", portanto durante o processo de criptografia é utilizado algum método que torna o que foi escrito ilegível para a maioria das pessoas. Os métodos de criptografia mais utilizados atualmente fazem uso de um algoritmo criptográfico, que por sua vez utiliza de chaves criptográficas para proteger a informação contra acessos indevidos. Os algoritmos criptográficos normalmente são de código fonte aberto, pois o segredo da criptografia não está no algoritmo, mas sim na chave criptográfica, que dessa forma deve ser muito bem guardada. Neste cenário a chave criptográfica é uma espécie de senha que somada ao algoritmo criptográfico irá cifrar ou decifrar a informação. Existem dois métodos de chaves criptográficas: chave única e chaves pública/privada. No método que utiliza chave única, conhecido também como chave simétrica, uma mesma chave é utilizada para cifrar e decifrar os dados. No outro método, a chave pública é utilizada para cifrar e somente a chave privada de uma dada chave pública é que pode decifrar as informações, tais chaves também são chamadas de assimétricas (Stallings e Brown, 2014);

- *Firewall*: Dispositivo que regula o que é ou não permitido geralmente fundamentado por alguma política de segurança (Bishop, 2004; Stallings e Brown, 2014). O tipo de *firewall* mais comum é o de filtro de pacotes de redes (Tanenbaum e Wetherall, 2013), que geralmente fica nas bordas das redes ligando dois ou mais segmentos, tal como LAN e Internet.

  Um *firewall* desse tipo trabalha normalmente até a camada de transporte do modelo TCP/IP (*Transmission Control Protocol/Internet Protocol*). Isso significa que as regras que regem o *firewall* podem ser criadas levando em consideração informações que identificam: (i) *Hosts*/redes de origem/destino, através das informações contidas no *datagrama* IP; (ii) Serviços de rede

(HTTP, FTP (*File Transfer Protocol*), DNS, etc), identificados pelos números de portas contidos nos protocolos da camada de transporte (TCP e UDP (*User Datagram Protocol*)). Também, alguns *firewalls* conseguem criar regras que levam em consideração os estados das conexões TCP.

- *Proxy*: Tipo de servidor de rede que faz intermédio entre a comunicação de dispositivos de redes (Bishop, 2004). Um *proxy*, no sentido literal da palavra, é um procurador que acessará serviços de rede em nome de seus clientes. Em ambientes de rede que utilizam *proxy* a comunicação não ocorre diretamente entre dois *hosts*, mas sim por intermédio do *proxy*, portanto o pacote que chega ao destino não é nunca o pacote original, mas sim um pacote totalmente[5] recriado pelo *proxy*, desta forma a resposta sempre será destinada ao *proxy*, que então poderá tratá-la e entregá-la ao cliente solicitante. Então, com o *proxy* cria-se um ponto de conexão, que não é comum em redes comutadas por pacotes (Tanenbaum e Wetherall, 2013), tal como redes TCP/IP. Devido a esse fato, torna-se possível, remontar todo o conteúdo de páginas Web e verificar se essa está de acordo com a política de segurança, já que o *proxy* consegue entender informações na camada de aplicação. O *proxy* também pode ajudar a manter a segurança implementando controle de acesso (usuário e senha), realizando registros de acesso à rede (*logs*) e ocultando a topologia da rede, já que é o *proxy* quem acessa a rede externa e não o cliente;

- IDS: Sistema que deve identificar problemas de segurança que estejam acontecendo. O comportamento padrão de um IDS, ao identificar um problema, é gerar um alerta para o administrador do sistema (Bashir e Chachoo, 2014; Sabahi e Movaghar, 2008). Há basicamente dois tipos de IDS: rede e *host*. Os baseados em *host* são conhecidos como HIDS (*Host Intrusion Detection Systems*) e têm por objetivo monitorar e detectar ações incomuns ou maliciosas que estejam ocorrendo localmente em um único *host*. Um HIDS pode monitorar recursos como: sistema de arquivos, processos em execução, arquivos de registro (*logs*), uso de CPU (*Central Processing Unit*) e memória, etc. O outro tipo de IDS é chamado de NIDS e tem por objetivo monitorar a segurança de redes de computadores. Um NIDS deve analisar os pacotes que passam pela rede em busca de problemas de segurança que possam afetar os computadores da rede. Tanto NIDS, como HIDS têm seus prós e contras, por exemplo: os NIDS conseguem monitorar várias máquinas ao mesmo tempo, porém não conseguem analisar pacotes que estejam criptografados (não há como analisá-los devidamente, já que algumas informações estão incompreensíveis), podem apresentar dificuldades em redes que utilizam *switches* (nesse ambiente os pacotes serão enviados apenas entre as portas do *switch* que ligam origem e destino, portando não necessariamente serão enviados para a análise do IDS, a menos que o *switch* possua recursos para isso, tal como replicação de portas – *mirror*) e também pode ocorrer de pacotes maliciosos passarem despercebidos se a rede analisada tiver um alto fluxo de pacotes, o que pode sobrecarregar o NIDS. Já um HIDS não possui problemas com *switches*, nem com pacotes criptografados, além de conseguir constatar se o ataque foi realmente efetivado, o que pode ser complicado para um NIDS, já que esse só observa o pacote passando pela rede e não sabe se o mesmo foi executado pelo alvo, todavia em grandes redes torna-se difícil manter os HIDS, já que é necessário instalá-lo em todos os computadores (Bishop, 2004; Stallings e Brown, 2014);

- IPS (*Intrusion Prevention Systems*): Tem o mesmo objetivo do IDS, mas um IPS além de identificar ataques, deve reagir tentando combatê-los (Nalavade e Meshram, 2010; Stallings e Brown , 2014). É comum utilizar o termo IDS para se referir a IPS, também existe a sigla IDPS (*Intrusion Detection Prevention Systems*) que faz menção a sistemas de segurança que têm a capacidade de prevenir problemas de segurança através da identificação de ataques a segurança bem como reagir a esses ataques (Patel *et al.*, 2012; Zargar *et al.*, 2013);

[5] O que é diferente de NAT (*Network Address Translation*), que altera apenas endereços IPs de origem/destino e portas (Tanenbaum e Wetherall, 2013)

- Anti-vírus: Um anti-vírus é bem similar a um IDPS, só que seu objetivo é monitorar e reagir a problemas relacionados a *malwares* (vírus, *worms*, cavalos de Tróia, etc) que possam afetar um computador. A tarefa básica de um anti-vírus é detectar e remover arquivos contaminados por vírus (Chang *et al.*, 2013; Egele *et al.*, 2008; Sanok, 2005);
- *Honeypot*/*Honeynet*: *Honeypot* é um sistema construído e preparado para ser invadido e tem por objetivo servir de isca para atacantes, de forma que pesquisadores estudem o comportamento do sistema e as técnicas empregadas durante ataques a segurança. Ele também pode ser utilizado para desviar o foco do atacante do ambiente real para um ambiente falso. O *Honeypot* é um ambiente computacional falso que pode ser implementado para simular partes de um sistema computacional. Com o advento da virtualização é comum ter *Honeypots* que representam serviços reais, com sistemas operacionais de verdade, mas que nunca possuem as informações reais que são desejadas pelos atacantes, assim os *Honeypots* podem simular ambientes tão reais e complexos quanto uma rede e aí passam a se chamar *Honeynets* (Bishop, 2004; Mairh *et al.*, 2011).

Mecanismos de segurança detectivos, como anti-vírus, IDS e IPS utilizam basicamente duas técnicas para realizar a identificação de problemas de segurança, que são métodos baseados em assinatura e por comportamento (Bashir e Chachoo, 2014; Bishop, 2004).

Mecanismos de detecção baseados em assinatura fazem uso de alguma característica marcante para identificar a ameaça à segurança (Sabahi e Movaghar, 2008). Por exemplo, para se identificar um vírus é possível pegar um trecho de código que representa o vírus e buscar o mesmo trecho em outros arquivos, se algum arquivo contiver o mesmo trecho de código, então ele está infectado. Nesse caso, a esse trecho de código que representa o vírus, se dá o nome de assinatura (Sanok, 2005). Então, geralmente um anti-vírus, assim como qualquer sistema baseado em assinatura, possui uma base de dados composta pelo maior número de assinaturas possível, pois são essas que irão detectar a ameaça (Egele *et al.*, 2008). No caso de NIDS, a assinatura poderá ser alguma informação contida nos pacotes de rede, tal como os caracteres que formam um comando malicioso ou uma combinação de parâmetros em protocolos/serviços que normalmente são utilizados em ataques (Bishop, 2004).

No caso de sistemas de detecção baseados em comportamento são utilizadas algumas heurísticas para identificar ataques a segurança, como analisar um conjunto de características e comparar com um conjunto de valores esperados (Bishop, 2004). Exemplificando, um NIDS pode identificar o comportamento padrão (o que é normal) em uma dada rede e tudo o que fugir desse padrão pode ser considerado uma ameaça. Esse tipo de técnica é chamada de detecção de anomalia (Bashir e Chachoo, 2014). No caso de anti-vírus, é possível identificar alguns *malwares* observando-se os arquivos que o software malicioso altera/utiliza, isto é, o sistema observa que o comportamento de um dado vírus é alterar/acessar sempre alguns determinados arquivos. Dessa forma, o anti-vírus fica monitorando os softwares em execução e caso algum apresente esse comportamento, o mesmo é identificado como vírus e pode ser bloqueado (Sanok, 2005).

Por fim, os mecanismos e métodos de segurança apresentados nessa seção podem ajudar muito na manutenção da segurança de ambientes computacionais, mas precisam ser bem avaliados pela equipe de TI, pois se forem utilizados incorretamente podem também trazer problemas, como passar a falsa impressão de que o sistema está seguro, o que é extremamente perigoso. A próxima seção discute alguns problemas dos mecanismos de segurança atuais e apresenta alguns desafios na área de segurança de computadores.

2.1.3 Desafios na área da segurança da informação

As seções anteriores apresentaram problemas de segurança já conhecidos, bem como mecanismos consolidados para prover segurança em ambientes informatizados. Contudo, um dos maiores desafios na área de segurança de computadores é o fato de que novas ameaças surgem frequentemente, pois as tecnologias evoluem em uma velocidade frenética. O mais preocupante é que os problemas tendem a seguir esse mesmo ritmo, mas em contrapartida os mecanismos de segurança dificilmente conseguem evoluir na mesma proporção.

Uma das maiores inovações para a segurança de computadores foi os mecanismos de segurança detectivos, pois esses conseguem identificar automaticamente problemas de segurança. Tais mecanismos são de grande ajuda, visto que a detecção de problemas em computadores é humanamente impossível na maioria das vezes. Contudo os mecanismos de segurança detectivos possuem dois grandes problemas, que são falsos positivos e falsos negativos.

Um falso positivo é quando o mecanismo de segurança detecta erradamente uma ação como sendo maliciosa, isto é, detecta um problema que não existe. Esse problema é complicado por inúmeros motivos, sendo alguns: (1) Se o mecanismo de segurança ficar alertando a respeito de problemas que não existem, os administradores do sistema poderão perder a confiança no mecanismo de segurança e consequentemente ele perderá sua utilidade. Por exemplo, pode-se chegar ao caso extremo do usuário ignorar um alerta correto a respeito de um problema que realmente esteja acontecendo, devido ao histórico de falsos positivos; (2) Se for um mecanismo que gera alertas de segurança, tal como um IDS, mensagens de alertas importantes podem ser ofuscadas, se o mecanismo gerar uma grande quantidade de mensagens a respeito de problemas que não existem, ou seja, as mensagens de falsos positivos podem atrapalhar a identificação e leitura de alertas reais; (3) Caso o usuário acredite que o alerta falso é real, isto é, representa uma ameaça verdadeira, esse pode tomar atitudes drásticas, tal como, apagar um arquivo idôneo no caso de um anti-vírus; (4) Seguindo a linha do item anterior, o sistema pode bloquear automaticamente ações legítimas dos usuários durante a ocorrência de falsos positivos, se forem utilizados mecanismos como um IPS.

Um falso negativo se caracteriza quando o mecanismo deixa de detectar um problema de segurança que realmente está acontecendo. O falso negativo - em muitos casos - é um problema mais sério que o falso positivo, pois quando ocorre um falso negativo significa que o mecanismo de segurança falhou e deixou que ações maliciosas passassem despercebidas, comprometendo a segurança do sistema. Durante um falso negativo, como o sistema não alerta a respeito do problema que está ocorrendo, o usuário fica com uma falsa sensação de segurança, quando na verdade o sistema já teve sua segurança corrompida.

É possível relacionar os problemas de falso positivo e negativo com o método de detecção utilizado, ou seja, detecção baseada em assinatura ou comportamento. Mecanismos de detecção baseados em assinatura podem sofrer com o problema de falso negativo se a base de assinaturas não estiver atualizada. Por exemplo, existe grande chance da assinatura de um vírus extremamente novo não estar na base de dados de muitos anti-vírus, dessa forma um anti-vírus sem a assinatura do novo vírus pode sofrer um falso negativo com essa vulnerabilidade dia-zero (Symantec, 2014). Técnicas de detecção baseadas em comportamento, como detecção de anomalia, podem apresentar bons resultados em relação à detecção de novos problemas de segurança, pois provavelmente novas ameaças irão apresentar um comportamento anormal (novo/diferente) durante o ataque e por isso serão detectados pelo mecanismo de anomalia. Todavia, mecanismos baseados em comportamento podem apresentar muitos problemas com falsos positivos. Exemplificando, um NIDS pode entender que um uso abrupto da rede foi gerado por uma ameaça, quando na verdade é uma ação legítima de um usuário que estava, por exemplo, fazendo cópias de segurança do sistema via rede.

Apesar de existirem várias pesquisas visando a solução de falsos positivos e negativos, na prática a melhor opção é manter os mecanismos de detecção sempre atualizados e corretamente configurados. Mesmo assim tais problemas continuam ocorrendo principalmente por depender de configurações/atualizações realizadas por administradores humanos que podem se esquecer de realizar as tarefas ou não ter expertise para configurar o mecanismo da maneira apropriada. Assim, as ameaças de falso positivo e negativo são problemas que estão em aberto dentro da área de segurança da informação e merecem atenção da comunidade científica.

Os mecanismos e ferramentas de segurança não são perfeitos e portanto possuem problemas, tais como os de falso positivo e negativo, descritos anteriormente. De fato, cada mecanismo de segurança tem suas limitações e são desenvolvidos para tratar de problemas específicos em contextos geralmente bem definidos, ou seja, um mecanismo de segurança normalmente não trata de uma vasta gama de problemas de segurança, exigindo assim que administradores façam uso de vários mecanismos de segurança, o que pode aumentar ainda mais a complexidade do ambiente.

Uma maneira de constatar as limitações dos mecanismos de segurança e obter um panorama geral a respeito das ameaças à segurança em ambientes informatizados, é observando os dados divulgados em relatórios anuais por empresas ligadas a área de segurança de computadores. A partir desses relatórios é possível observar que atualmente:

- As principais vulnerabilidades exploradas em ambientes corporativos e domésticos são relativas a: navegadores Web (*browsers*), Android, Java, Microsoft Office, Adobe Flash Player e Adobe Reader (Kaspersky, 2013, 2015);

- 97% das aplicações analisadas pela Trustware (2016) possuíam uma ou mais falhas graves, que podem ser exploradas em ataques à segurança. As falhas mais comuns são de *cross-site scripting*, vazamento de informações, autenticação, autorização e gerenciamento de sessões. Segundo dados da Symantec (2014) entre os anos de 2006 a 2013, foram exploradas em média 5.397 vulnerabilidades por ano, sendo que 13,5% em média eram vulnerabilidades dia-zero. Em 2015 foram descobertas 54 vulnerabilidade dia-zero (Symantec, 2015);

- Os métodos mais empregados para propagação de *malwares* utilizam redes de computadores, principalmente tecnologias como: programas de mensagens instantâneas, *e-mail* via SMTP (*Simple Mail Transfer Protocol*), CIFS (*Common Internet File System*), programas de transferência de arquivos P2P e atualmente o sistema operacional Android (Symantec, 2014, 2015);

- Com relação aos métodos de invasão, de acordo com a Trustware (2016), os ataques normalmente são: 13% acessos remotos, 12% injeção de SQL (*Structured Query Language*), 12% erros de configuração, 10% relacionadas a falhas que permitem que o acatante faça indevidamente *upload* de arquivos (*file upload flaw*), 8% engenharia social e *phishing*, 7% ameaças internas, 7% injeção de códigos maliciosos, 7% vulnerabilidades em aplicações de servidores e 7% senhas fracas, por fim 17% dos métodos de ataques são desconhecidos;

- O relatório da Sophos (2014a) informa que o uso de *botnets* cresceu nos últimos anos, principalmente devido ao fato do vazamento do código fonte da *botnet* Zeus, o que permitiu que fossem criadas novas tecnologias de controle de *botnets*, mecanismos de criptografia, redundância dos canais de comunicação, entre outras inovações que também ajudaram a aumentar o número de ataques DDoS. Ainda segundo o relatório, um ataque de uma *botnet* chegou a mobilizar 500.000 zumbis para realizar um dado ataque. Geralmente os ataques com *botnets* tem como alvo grandes corporações, mas afetam pequenas empresas e indivíduos, durante os ataques (Kaspersky, 2013). No ano de 2015 o número de *bots* diminuiu em alguns países, mas em contrapartida na china esse número cresceu 84% (Symantec, 2015).

- O número de ataques DoS têm amentado a cada ano (Cisco, 2014, 2016; Symantec, 2014, 2015). Segundo Symantec (2014), alguns ataques modernos chegam a direcionar cerca de 300Gbps de pacotes maliciosos para uma vítima. Isso ocorre por que os ataques estão mais elaborados com o uso de *botnets* e curiosamente com a possibilidade de utilizar novamente técnicas de ataques DoS empregadas no passado. As técnicas de DoS do passado voltaram a surtir efeito devido ao aumento do número de computadores conectados a Internet através de acesso via banda larga, o que colocou clientes com péssimas ou nenhuma configuração de segurança conectados às redes;

- Ataques internos são os mais devastadores, um excelente exemplo é o caso da exposição de dados da NSA (*National Security Agency*) feita por Edward Snowden, um dos agentes da própria agência (Starr e Yan, 2013). Além, dos ataques internos propositais também existem os problemas de segurança internos acidentais. Conforme a Symantec (2014), as empresas brasileiras são as mais afetadas por erros humanos;

- 59% das vítimas de ataques de segurança não conseguiram detectar os problemas sozinhos. Mas, quando as empresas/pessoas possuem capacidade de se auto-proteger, essas demoram

em média 15 dias para detectar o problema e mais 1 dia para contê-lo. Já quando a detecção e reação aos problemas de segurança depende de terceiros (empresas de segurança), leva-se 168 dias do inicio do ataque até a sua detecção, e adicionalmente 28 dias para remediá-lo (Trustware, 2016);

- Dados do relatório da Symantec (2014, 2015) informam que a maioria das vulnerabilidades exploradas por ataques não são recentes, tal como vulnerabilidades dia-zero. Pelo contrário a maior parcela das vulnerabilidades exploradas, são conhecidas a muitos anos, bem como têm sua correção disponível também a muito tempo. Todavia as vítimas não corrigem os problemas ou sequer aplicam as correções disponíveis, devido principalmente a falta de capacidade técnica, ignorância a respeito do problema e até por questões sociais/financeiras, tal como o uso de softwares não licenciados.

Usando como base os dados apresentados pelos relatórios feitos pelas empresas de segurança é possível constatar que muitos ataques têm ligação com as redes de computadores. Na maioria das vezes os ataques utilizam as redes de computadores para iniciar suas atividades, alguns ataques são sustentados via rede, como por meio do envio de dados das vítimas ou permitindo que o atacante controle o computador comprometido, e também muitas das pragas digitais têm por finalidade comprometer recursos computacionais ligados às redes de computadores.

Os relatórios de segurança enaltecem o crescimento dos problemas de segurança direcionados a empresas e nos últimos tempos também às pessoas comuns. Isso se dá na maioria dos casos devido ao fato desses estarem conectados por mais tempo na Internet. Outro ponto importante é que na maioria dos casos tanto empresas quanto indivíduos não são autossuficientes na resolução de problemas de segurança e por isso normalmente dependem de terceiros, todavia algumas soluções de terceiros podem apresentar um retardo muito grande em relação ao problema, o que deixa uma janela de tempo para ataques as essas vulnerabilidades.

Além do aumento no número de ataques a segurança, da incapacidade dos usuários em combater as ameaças em tempo hábil e de maneira eficaz, os ataques estão ficando cada vez mais sofisticados, o que agrava o problema. Um exemplo de ataque sofisticado é o realizado pelo BlackHole, que é um *exploit kit*, isto é um conjunto de ferramentas de ataques que visa comprometer a segurança de sistemas computacionais explorando vários tipos de vulnerabilidades. Os ataques utilizando o BlackHole iniciam geralmente através do envio de *e-mails* (*spams*) com conteúdos falsos, que utilizam técnicas de engenharia social objetivando levar o leitor do *e-mail* a acessar alguma página Web maliciosa. Quando a vítima acessa a página Web maligna, o BlackHole começa a procurar por informações a respeito dos softwares utilizados na máquina da vítima, tais informações normalmente são algo como: tipo de software para leitura de PDF (*Portable Document Format*), navegador Web utilizado, versões do Adobe Flash Player, versões do Java, sistema operacional, etc. De posse das informações a respeito dos softwares e sistema operacional da vítima, o BlackHole escolhe qual desses é mais fácil de comprometer. Feita a escolha, o BlackHole envia um *malware* que pode executar várias ações a mando do atacante, sendo que o mais comum é redirecionar a vítima para páginas Web falsas, para que principalmente, o atacante se passe por entidades financeiras para roubar dados a respeito de contas bancárias da vítima. O BlackHole permite inclusive que o atacante acompanhe o andamento dos ataques e de informações das vítimas em uma interface Web bem amigável. O BlackHole é largamente utilizado na realização de cibercrimes (crimes cibernéticos) e o seu desenvolvimento tem fins lucrativos. Dessa forma, para o atacante utilizar o BlackHole é necessário pagar por uma "licença" de uso. Atualmente os valores, em dólar, são algo em torno de: $50 por dia, $200 por semana, $500 por mês ou $1.500 por ano, o que demostra que cibercrimes são altamente rentáveis (Kaspersky, 2012; Sophos, 2014b). Conforme a Trustware (2014), dentre os *exploit kits*, o BlackHole foi o mais utilizado nos últimos anos.

Além dos ataques estarem mais sofisticados, os ambientes computacionais também estão se tornando mais complexos. Um exemplo, é um fenômeno atual chamado BYOD, que consiste em permitir que funcionários levem seus dispositivos móveis pessoais e utilizem dentro das empresas (Disterer e Kleiner, 2013; Miller *et al.*, 2012). Através de um *tablet* ou de um *smartphone* do próprio

funcionário, é possível acessar os recursos computacionais da empresa, tal como, Internet, softwares ou os bancos de dados. Também, não é incomum que os funcionários levem os computadores corporativos para fora da empresa, tal como, em visitas a fornecedores, clientes ou mesmo para a casa do próprio funcionário. Outra prática comum atualmente é o uso de computação em nuvem (*cloud computing*) (Winkler, 2011), isso ocorre devido ao fato das informações armazenadas em ambientes de nuvem tornarem-se altamente disponíveis, com a utilização de sistemas distribuídos.

Dá-se o nome de relação *any-to-any* (qualquer-para-qualquer) (Cisco, 2014) a todas as modalidades de acesso a rede e aos dados, descritas no parágrafo anterior. O nome vem do fato de que qualquer dispositivo em qualquer local pode acessar qualquer aspecto da rede, ou seja, dispositivos pessoais acessam a rede empresarial ou dispositivos empresariais utilizam os recursos de qualquer outra rede, tal como, das casas dos funcionários, parceiros de negócios ou redes desconhecidas.

Em um ambiente tão amigável e permissível como o de BYOD fica mais complicado de se manter a segurança, devido principalmente ao fato dos funcionários terem acesso/permissão total aos seus equipamentos pessoais. Assim, eles podem instalar qualquer tipo de software, bem como acessar qualquer tipo de conteúdo na Internet, (Miller *et al.*, 2012; Scarfo, 2012) o que aumenta a chance desses dispositivos contraírem alguma vulnerabilidade, que poderá ser explorada dentro da empresa. Para exemplificar, os dispositivos normalmente utilizados na prática de BYOD são móveis (*laptops*, *smartphones* e *tablets*) e conforme a Kaspersky (2013), os *malwares* para esses dispositivos começaram a ser muito explorados no ano de 2011 e até o final de 2013 foram contabilizados mais de 148.427 *malwares* para tais equipamentos. Esse grande número de pragas digitais demostra o interesse dos atacantes com esse tipo de dispositivo, sendo que 99% dos *malwares* são destinados para a plataforma Android (Cisco, 2014). Contudo, proibir a prática do BYOD para garantir a segurança não é indicada, pois muitos indivíduos possuem ou possuirão dispositivos móveis e tentarão acessar pelo menos a rede da empresa. Assim, são necessários novos mecanismos de segurança para não transformar o BYOD ironicamente em BYOT (*Bring Your Own Threat*) - traga sua própria ameaça/vulnerabilidade (Anderson, 2012; Scarfo, 2012).

Quanto ao caso dos dispositivos das empresas acessarem redes externas, isso também é perigoso, pois as redes em sua grande maioria estão expostas a problemas de segurança. Por exemplo, 100% das redes analisadas pela Cisco (2014), apresentam tráfego de rede indo para páginas Web que possuem *malwares*. Dessa forma, quando tais dispositivos voltarem para dentro da empresa esses podem contaminar a infraestrutura da mesma.

O uso de computação em nuvem é muito interessante em vários aspectos, principalmente para manter a informação mais disponível. Todavia essa disponibilidade pode comprometer a confidencialidade das informações, já que normalmente em ambientes de nuvem os dados ficam espalhados em vários computadores/servidores, o que torna difícil saber quem (administrador ou atacante) tem acesso às informações e se esses são confiáveis ou não (Winkler, 2011). O caso Edward Snowden e NSA demonstra que há muita gente interessada nos dados que trafegam ou, nesse caso, são armazenados na Internet (Starr e Yan, 2013).

Neste sentido é muito difícil saber quais relacionamentos de rede são confiáveis e quais não são, o que pode tornar a facilitadora estrutura de acesso *any-to-any* em "problemas de segurança *any-to-any*" (Cisco, 2014), já que a ameaça pode surgir em qualquer ponto da estrutura do ambiente computacional da empresa, principalmente por existirem partes da estrutura que estão além das possibilidades de controle da equipe de TI da empresa, como é o caso das nuvens e dos dispositivos pessoais dos funcionários. Todavia esses problemas devem ser superados, pois a sociedade contemporânea anseia cada vez mais por recursos computacionais que permitam a integração de trabalho e diversão da forma mais rápida, amigável e segura possível.

Esse trabalho propõem uma arquitetura autonômica que auxilie na segurança dos ambientes modernos de redes de computadores. Para isso, a próxima seção (2.2) desse capítulo, apresenta os conceitos de Computação Autonômica que serão utilizados para construir uma arquitetura de segurança de redes que dependa minimamente dos usuários ou administradores da rede, já que esses muitas vezes não conseguem lidar com os problemas de segurança da maneira devida, tal como como foi abordado na presente seção. A seção 2.3 desse capítulo, apresenta a tecnologia de Redes Definidas

por Software, que permitirá programar a rede para que essa detecte e reaja autonomicamente aos problemas de segurança, objetivando solucioná-los ou mitigá-los.

2.2 Computação autonômica

Os sistemas computacionais estão ficando cada vez mais complexos e tornando-se mais difíceis de serem gerenciados por humanos (Kephart e Chess, 2003). Um problema similar já ocorreu com a telefonia em 1920, mas esse problema foi resolvido através da automatização do sistema telefônico. Da mesma forma, uma possível solução para se manter os sistemas computacionais modernos sempre ativos e em bom funcionamento é utilizando técnicas autonômicas de tarefas de gerenciamento/manutenção desses ambientes (Huebscher e McCann, 2008). A CA, do inglês *Autonomic Computing*, é uma tecnologia que visa diminuir a interação do ser humano com os computadores no aspecto da manutenção e gerenciamento dos equipamentos (Huebscher e McCann, 2008; Kephart e Chess, 2003).

A CA é inspirada na biologia, mais especificamente na ideia de que partes do corpo de seres vivos trabalham de forma independente. Um termo que simboliza bem o que se espera da CA é homeostase, que representa a habilidade de sistemas complexos se adaptarem ao ambiente interno e externo para conseguir sobreviver. A adaptação dos sistemas é obtida através do equilíbrio homeostático, tal equilíbrio é mantido por meio de vários processos de retroação, que ajustam os sistemas visando atingir um ponto de equilíbrio que nunca é atingido, pois o ambiente onde reside o sistema é aberto e está em constante modificação. Assim, um sistema homeostático utiliza autorregulação fazendo com que o organismo corrija parâmetros mantendo sua integridade (Hariri *et al.*, 2006). A grande maioria dos órgãos internos dos seres vivos trabalham de forma autônoma, não sendo necessário que o ser vivo pense em um órgão específico para que esse funcione. Por exemplo, não é necessário lembrar do coração para que ele desempenhe sua função e bombeie sangue para o restante do corpo, ou seja, ele trabalha de forma independente (Huebscher e McCann, 2008). Então, espera-se com o uso da computação autonômica que partes dos sistemas computacionais trabalhem de maneira independente e autônoma, sem a supervisão de administradores. Com o uso de técnicas de CA espera-se que um computador ou partes desse funcionem sem intervenção direta do administrador do sistema.

O conceito de CA foi introduzido pela IBM (*International Business Machines*) no ano de 2001 e se referia a sistemas de computadores auto-gerenciáveis (*self-managing*) (Kephart e Chess, 2003). A IBM apresentou essa ideia com o intuito de retirar parte do ônus dos administradores de sistemas computacionais complexos através da criação de técnicas que permitam o auto-gerenciamento, manutenção e otimização desses computadores, diminuindo a interação humana na gerência desses ambientes.

2.2.1 Propriedades da computação autonômica

Para utilizar CA são requeridas algumas propriedades do sistema computacional. Porém antes, faz-se necessário definir o que é "sistema" no contexto de CA. Segundo Kephart e Chess (2003) sistema é um conjunto de recursos computacionais interligados para executar uma ou mais funções específicas. Portanto para CA um microprocessador pode ser visto como um sistema de mais baixo nível, já um computador composto de vários processadores, representa outro nível de sistema, e por sua vez computadores interligados para prover serviços a clientes podem ser vistos como um sistema de mais alto nível. A CA pode ser aplicada em todos níveis de sistema - hardware e software - com o intuito de fazer os possíveis sistemas funcionarem de forma independente, mas trabalhando para um objetivo comum.

Ainda conforme Kephart e Chess (2003) no manifesto da IBM um sistema autonômico deve possuir as seguintes propriedades:

- O sistema deve conhecer a si próprio (*self-awareness*) e compreender outros componentes que também representam sistemas. Como um sistema pode ter vários níveis, ele precisa conhecer

todos os detalhes de seus componentes e todas as conexões com outros sistemas para poder administrar a si mesmo (*self-managing*);

- A configuração do sistema deve ocorrer automaticamente, ou seja, o sistema tem que se auto-configurar (*self-configuring*). O ajuste de parâmetros do sistema tem que ser dinâmico e se auto adaptar para proporcionar o bom funcionamento;
- O sistema deve se auto-otimizar buscando sempre o melhor desempenho. Um sistema autonômico não pode se contentar com seu estado atual, tentando sempre a auto-otimização (*self-optimization*). Essa propriedade permitirá que demandas conflitantes de serviços sejam atendidas da melhor forma possível, podendo então priorizar tarefas e fornecer qualidade de serviço;
- O sistema autonômico pode ser capaz de se auto-regenerar ou auto-curar (*self-healing*) de falhas que venham a ocorrer. Deve ser possível detectar problemas e então achar formas de resolver ou minimizar o mesmo;
- Um ambiente autonômico deve ter capacidade de se auto-proteger (*self-protection*) de ataques e ameaças que possam afetar a segurança e integridade do sistema. Espera-se que sistemas autonômicos detectem o problema e reajam antecipadamente às ameaças retirando o sistema de situações de risco;
- Um sistema de computadores autonômicos atua segundo seu ambiente e contexto (*context aware*). Essa propriedade é parecida com a auto-otimização, só que fundamentada no que ocorre externamente ao sistema. O sistema autonômico tem que avaliar os recursos disponíveis e negociar o uso de elementos subutilizados, adaptando a si mesmo e o ambiente ao seu entorno. O ambiente é sensível ao contexto e fornece serviços baseados em conhecimentos sobre os estados das transações realizadas. Deve ser possível descobrir outros sistemas e recursos, da mesma forma o sistema deve conseguir se auto-descrever para outros sistemas;
- O sistema deve ser implementado através de padrões abertos e ter habilidade de funcionar em ambientes heterogêneos, ou seja, CA não pode ser uma solução proprietária;
- A CA deve ser transparente. O sistema pode antecipar, otimizar, configurar, gerenciar e proteger o usuário, entretanto o sistema não pode transparecer o que está ocorrendo e principalmente o usuário não precisa ter ciência da complexidade empregada na CA.

Huebscher e McCann (2008) consideram de extrema importância na CA as propriedades auto-X (do inglês, *self-X*): auto-configuração, auto-otimização, auto-cura e auto-proteção. Segundo Sterritt *et al.* (2005), com o crescimento do uso de CA foram surgindo novas propriedades auto-X, como: auto-antecipação, auto-adaptação, auto-definição, auto-destruição, auto-diagnóstico, auto-governança, auto-recuperação, auto-reflexão, auto-estabilização, auto-simulação, dentre outras. Sterritt *et al.* (2005) denomina auto-X como *selfware*, que é qualquer sistema que utiliza componentes de CA.

As propriedades auto-X foram inspiradas no trabalho de Wooldridge e Jennings (1994), que descrevem agentes no contexto de IA (Inteligência Artificial). O mesmo define que agentes fazem coisas, ou seja, atuam ou agem com seu ambiente. No contexto de CA os agentes são representados pelos sistemas. Agentes devem ter autonomia (*autonomy*), agindo sem controle ou supervisão humana. Os agentes também têm racionalidade (*rationality*), que pode ser entendida como a capacidade de maximizar sua performance através do estado atual do sistema. Um agente também tem capacidade de interagir com outros agentes ou com seu ambiente, o que Huebscher e McCann (2008) chamaram de habilidade social. Todos os itens citados estão relacionados intimamente com CA e são base para as propriedades auto-X. Existem discussões sobre o que é CA, pois muitos trabalhos utilizam as propriedades auto-X. Um sistema auto-adaptável (*self-adaptive*) segundo Macías-Escrivá *et al.* (2013) deve se adaptar ao ambiente e utiliza para isso pelo menos uma das propriedades auto-X, principalmente auto-otimização. Contudo, atualmente, a CA propriamente dita é caracterizada pelo uso de duas ou mais propriedades auto-X (Huebscher e McCann, 2008).

2.2.2 Níveis de automação

Como definido anteriormente, a CA tem por objetivo automatizar tarefas relativas ao gerenciamento e manutenção de computadores, diminuindo esforços humanos na administração de sistemas. Existem vários níveis de automatização para ambientes computacionais. Tomando como base os trabalhos IBM (2006) e Nami e Bertels (2007), identifica-se os seguintes níveis de automatização:

Básico: nesse nível não é utilizada nenhuma automação para ajudar no gerenciamento do sistema, as tarefas são feitas manualmente;

Gerenciável: automatiza o monitoramento do sistema. Utiliza tecnologias para coletar informações, auxiliando o administrador na análise do ambiente. Nesse nível começa a ser empregado algum tipo de automatização;

Preditivo: nesse nível é possível que o recurso computacional se auto-monitore, analise os dados obtidos e emita alertas, tornado o serviço do administrador de sistema mais confortável;

Adaptativo: aqui o recurso computacional pode se auto-gerenciar, realizando tarefas administrativas sem a necessidade de intervenção humana. IBM (2006) chama esse nível de ciclo fechado, pois a automatização do gerenciamento é possível através do ciclo: monitoramento, análise, planejamento e execução, que será melhor detalhado na seção 2.2.3. O ciclo tem por objetivo manter políticas pré-estabelecidas pelo administrador do sistema;

Autonômico: esse nível é mais complexo que o anterior, pois implementa automação em todo ambiente computacional, retirando ao máximo a tarefa administrativa das mãos humanas. O nível anterior contempla mais a automação local ou de um único recurso, já essa é de âmbito global e deve permitir que toda estrutura computacional trabalhe de forma independente e autonômica. O que deve permitir que a equipe de TI se concentre apenas no processo de negócio e não mais na manutenção do ambiente computacional.

2.2.3 Arquiteturas de computação autonômica

2.2.3.1 Arquitetura de Ashby

Sistemas computacionais modernos são compostos por inúmeras variáveis. Para que um sistema complexo sobreviva é necessário manter limites para certas variáveis essenciais. Segundo Hariri *et al.* (2006), a perda do limiar das variáveis pode ser catastrófica para alguns sistemas. Ashby (1960) estudou mecanismos internos do corpo humano que trabalham para manter tais limites e formulou o sistema ultra-estável de Ashby, que foi revisado por Hariri *et al.* (2006) para ter uma conotação mais computacional.

Ashby (1960) notou que sistemas normalmente sofrem com pequenas alterações nas variáveis principais e eventualmente distúrbios maiores em seus parâmetros. Inspirado nisso foi desenvolvida uma arquitetura que consiste em dois laços de repetição, sendo que um laço controla pequenos distúrbios e outro grandes alterações. A arquitetura ultra-estável de Ashby é apresentada na Figura 2.1 e é composta por:

- Ambiente, que é o lugar ou contexto no qual o sistema reside; As variáveis essenciais, que representam o estado do sistema;
- O mecanismo de reação "R", que executa ações visando manter o equilíbrio do sistema;
- O canal motor, que é utilizado pelo mecanismo de reação para efetuar mudanças internas ou externas;
- Canal sensor, que é responsável por perceber alterações no ambiente interno ou externo;
- Mecanismo de passos "S", que analisa parâmetros que provocam mudanças em características relevantes do sistema.

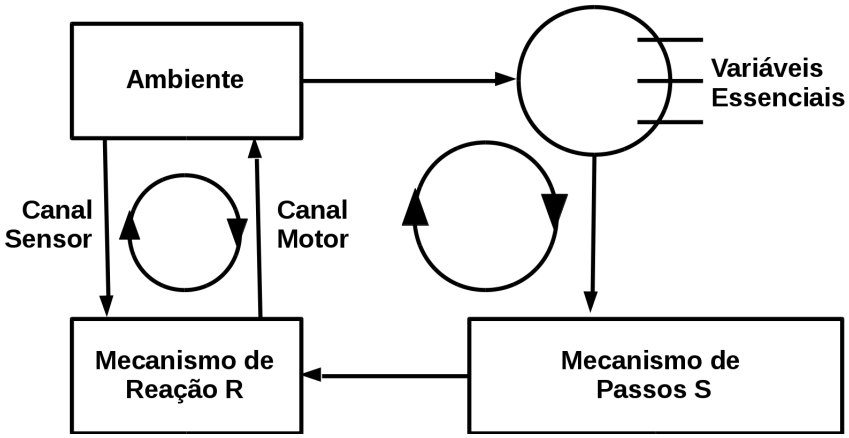


Figura 2.1: *Arquitetura ultra-estável (Ashby, 1960)*

Nessa arquitetura existe comunicação retroativa entre o ambiente e "R". O subsistema "R" reage a pequenas mudanças visando manter os limites de forma que todo o sistema se adapte ao ambiente alterado. Quando há transformações bruscas, afetando as variáveis essenciais, a segunda comunicação retroativa é ativada, alterando o comportamento atual do sistema para adaptar-se às variações, isso é feito pelo "S" que é acionado através das varáveis essenciais. O "S" deve selecionar parâmetros que mantenham o equilíbrio de todo o sistema e traduzir tais parâmetros para o subsistema de reação "R", selecionando um novo conjunto de comportamento para que o sistema volte ao seu equilíbrio frente à mudanças drásticas no ambiente e assim sobreviva (Hariri *et al.*, 2006).

Hariri *et al.* (2006) apresenta uma arquitetura conceitual de CA baseada na arquitetura de Ashby (1960), que pode ser vista na Figura 2.2. O modelo consiste dos seguintes módulos:

1. **Ambiente computacional:** é a estrutura computacional composta de hardware e/ou software que será gerenciada;

2. **Ambiente:** representa todos os fatores internos e externos ao sistema que podem influenciar o equilíbrio do ambiente computacional;

3. **Controle:** gerencia alterações no sistema que ocorrem em tempo de execução. Esse módulo gerencia os objetivos do sistema com ajuda do laço de controle local e global, acionados por sensores de forma reativa ou proativa. O módulo controle possui ainda os seguintes submódulos:

   - **Monitoramento e análise (M&A)** - monitora e analisa o ambiente computacional. O monitoramento é feito com ajuda de **sensores**;
   - **Planejamento** - planeja estratégias de execução visando otimizar o ambiente computacional. A otimização pode ser qualquer propriedade auto-X, vista na seção 2.2.1;
   - **Mecanismo de conhecimento** - fornece suporte ao módulo controle para decidir qual regra, política ou ação é mais apropriada para melhorar a performance do sistema.

4. **Laço de controle local:** controla ou otimiza o comportamento dos elementos locais do sistema. Esse módulo só irá trabalhar com estados locais e com regras de comportamento presentes no mecanismo de conhecimento. O laço de controle local implementa auto-gerenciamento, só que em nível local e portanto não tem capacidade de gerenciar o sistema em um contexto geral, a ponto de tomar decisões ótimas para todo o sistema;

5. **Laço de controle global:** gerencia o comportamento global do sistema. Esse laço é disparado quando as variáveis essenciais do sistema atingirem valores inaceitáveis ou para otimização. O laço global pode trabalhar com valores desconhecidos e utiliza pontos cardeais (que podem

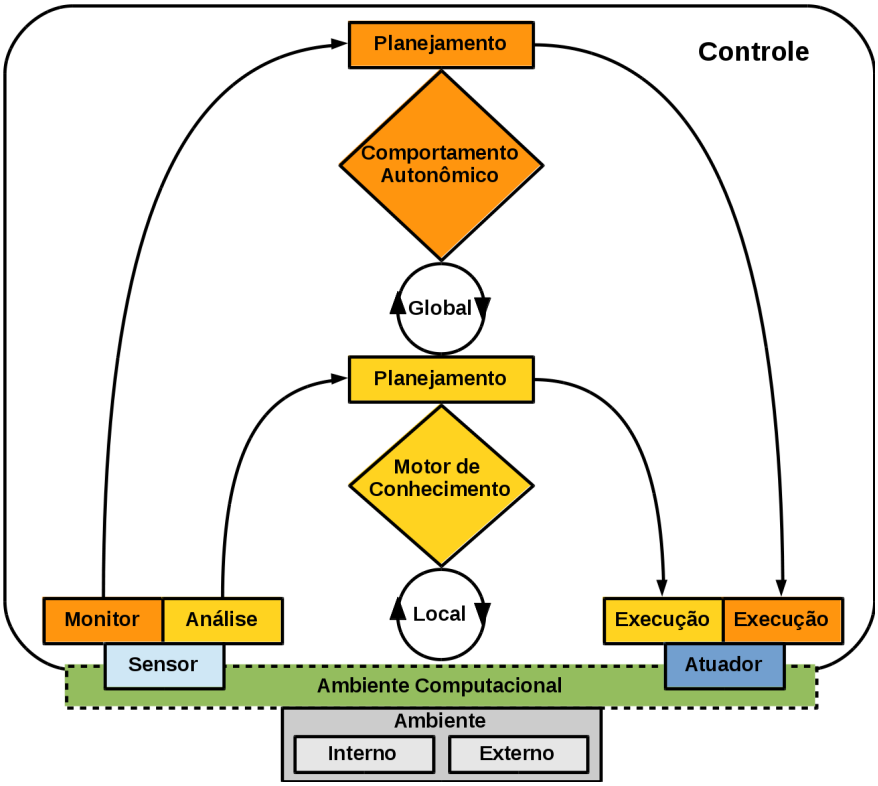


Figura 2.2: *Arquitetura da computação autonômica: visão conceitual (Hariri et al., 2006)*

ser, por exemplo, desempenho, tolerância a falhas, configuração ou segurança) para monitorar e analisar o ambiente computacional. O controle desse laço pode agir alterando o comportamento computacional, adaptando o mesmo ao ambiente atual. Isso normalmente é feito às cegas e pode degradar o desempenho do sistema. A ação a ser tomada pelo laço de controle global é escolhida dentre uma série de ações pré-definidas com a ajuda do submódulo de planejamento e do mecanismo de conhecimento;

6. **Execução:** aplica o novo plano de ações com objetivo de adaptar todo sistema às novas condições do ambiente. As ações são repassadas ao ambiente computacional com ajuda de atuadores.

2.2.3.2 MAPE-K

Para o desenvolvimento de sistemas que utilizam CA, a IBM propôs a arquitetura MAPE-K (*Monitor*, *Analyse*, *Plan*, *Execute and Knowledge*), que é derivada dos termos: monitoramento, analise, planejamento, execução e conhecimento, que dão origem ao nome da arquitetura e são conceitos chave da mesma (IBM, 2006). Essa arquitetura proposta pela IBM é muito utilizada por sistemas que implementam auto-gerenciamento (Huebscher e McCann, 2008; Teles *et al.*, 2011).

Segundo Huebscher e McCann (2008), a arquitetura MAPE-K é provavelmente inspirada no modelo de agente de IA proposto por Russell e Norvig (2003). A Figura 2.3 ilustra o modelo de Russell e Norvig (2003), sendo que o agente percebe o ambiente através de sensores e realiza ações utilizando atuadores. A interação entre o ambiente e o agente forma um ciclo/laço - agente influencia no ambiente e ambiente no agente, por isso a arquitetura MAPE-K também é chamada de ciclo MAPE-K autonômico.

O MAPE-K é constituído por um conjunto de elementos autonômicos que podem ser hardware ou software, gerenciados por um gerente autonômico. Visando proporcionar interação entre o elemento e o gerenciador existem sensores que coletam informações a respeito do elemento a ser gerenciado e os atuadores que executam ações, que são ordens do gerenciador para o elemento.

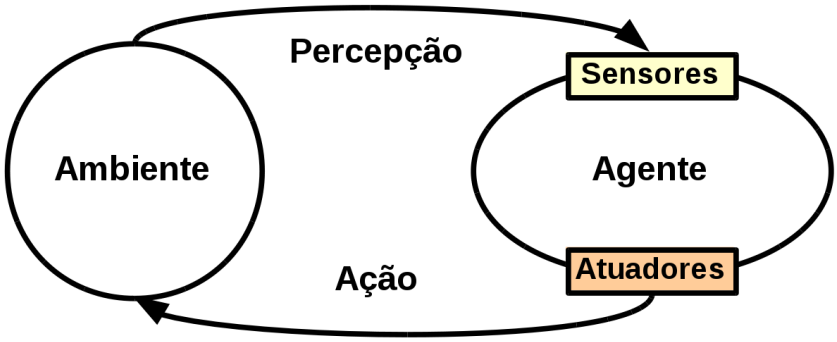


Figura 2.3: *Agente interagindo com o ambiente (Russell e Norvig, 2003)*

Um exemplo prático do uso da MAPE-K, pode ser: Imaginando uma aglomerado computacional (*cluster*) que fornece páginas *Web*. Tal aglomerado é composto por inúmeros servidores HTTP. Um dos objetivos do uso de CA nesse aglomerado é economizar recursos trabalhando sobre demanda, ou seja, aumentando o número de servidores quando existirem muitas requisições à páginas HTTP e desativando servidores conforme a demanda diminuir para, por exemplo, economizar energia. Empregando a arquitetura MAPE-K, os servidores são os elementos a serem gerenciados. Os sensores ficam monitorando o uso de recursos desses servidores, tal como, processador, memória, carga da rede, etc. As informações coletadas pelos sensores são analisadas pelo gerenciador autonômico que pode ligar mais servidores, desligar servidores quando os recursos estiverem subutilizados ou mesmo decidir por não fazer nada. A ação de desligar ou ligar servidores é realizada pelos atuadores. No cenário descrito, o gerenciador autonômico poderia aplicar outras propriedades da CA como auto-cura. Nesse caso, quando um servidor HTTP apresentasse falhas, as tarefas delegadas a ele poderiam ser atribuídas a outro servidor; ou poderia ser aplicada auto-otimização. Nesse caso, poderia-se manter os servidores sempre trabalhando em carga máxima a fim de evitar que novos servidores sejam ativados desnecessariamente.

A estrutura completa do ciclo MAPE-K autonômico pode ser vista na Figura 2.4. Sendo a arquitetura composta por:

Sensores: realizam coleta de informações do elemento gerenciado. As informações coletadas podem ser das mais diversas e variam de tipo conforme o elemento a ser observado;

Monitor: pode implementar mecanismos para filtrar as informações coletadas pelos sensores, isso é necessário para que as informações coletadas não sobrecarreguem o sistema causando algum tipo de DoS, o que afetaria toda a estrutura retroativa do MAPE-K. Existem vários tipos de monitoramento: passivo, ativo, orientado a eventos, orientado ao tempo, etc... (Teles *et al.*, 2011). O monitor deve selecionar dados que demostrem o comportamento do sistema atual, ou seja, informações que sejam úteis para realizar o auto-gerenciamento do elemento monitorado. Os dados selecionados pelo monitor devem ser armazenados e correlacionados até gerarem informações que identifiquem algum tipo de sintoma que precise ser analisado (IBM, 2006);

Análise: recebe informações do módulo monitor e tem como função analisar e encontrar situações que indiquem risco ao equilíbrio do sistema. Deve identificar sintomas que exijam mudanças imediatas ou futuras. O módulo tem que garantir o cumprimento dos objetivos especificados pelo administrador, caso os objetivos não estejam sendo atendidos deve requisitar ao módulo de planejamento mudanças que mantenham a integridade do sistema (IBM, 2006);

Planejamento: cria ou seleciona planos de ações fundamentados nos exames minuciosos realizados pelo módulo de análise visando o restabelecimento do equilíbrio do sistema. O plano pode ser um simples comando ou uma sequência complexa de passos (IBM, 2006). Os objetivos a

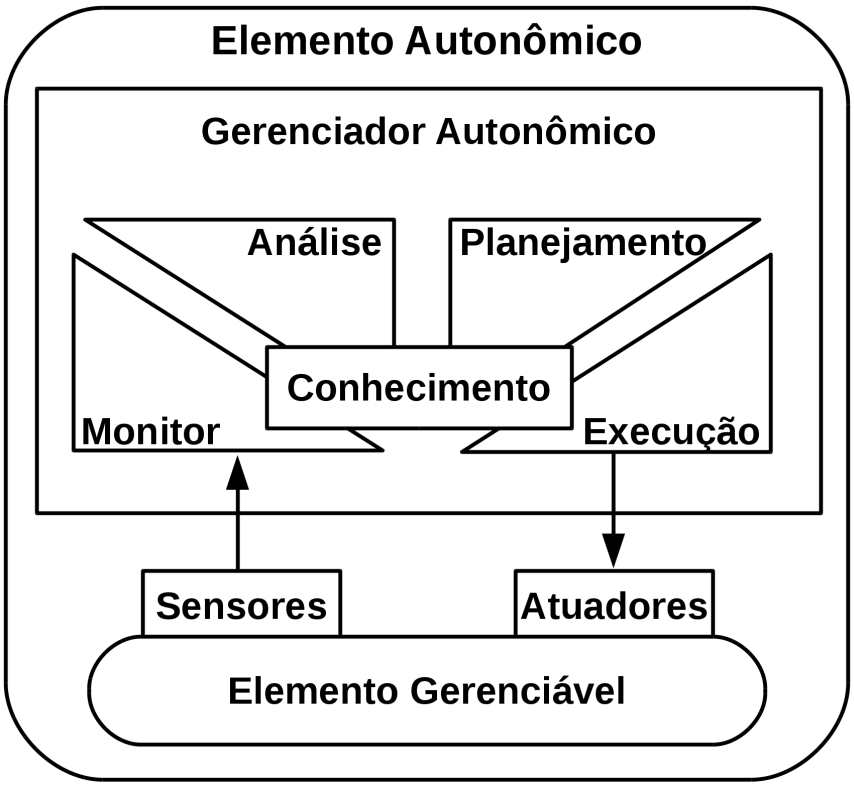


Figura 2.4: *Arquitetura de ciclo MAPE-K autonômico (IBM, 2006)*

serem atendidos pelo planejamento são representados através de políticas de ECA (Evento-Condição-Ação), políticas de meta (*goal policies*) ou políticas de função de utilidade (*utility function policies*) (Huebscher e McCann, 2008; Teles *et al.*, 2011; White *et al.*, 2004).

Execução: possui mecanismos para executar as mudanças e atender o plano de ação enviado pelo módulo de planejamento. Pode ser necessário executar uma ou mais ações no elemento gerenciado para conseguir mudar o estado do sistema e atingir o objetivo planejado. Parte da execução do plano de ação pode ser atualizar o módulo de conhecimento, que será descrito a seguir;

Atuadores: também chamados de efetores (*effectors*), são responsáveis por materializar e efetivamente executar as ações que foram determinadas pelo módulo de execução, para serem aplicadas no elemento gerenciado;

Base de conhecimento: é um repositório de dados, que permite através de uma interface da arquitetura, obter informações sobre históricos de conhecimento a respeito do sistema. As informações armazenadas podem ser: políticas do sistema, trocas de estado já realizadas, métricas, objetivos, planos de ações, problemas enfrentados, soluções e eficiência das mesmas, dentre outras. Esse conhecimento pode ser ainda compartilhado com outros gerentes autonômicos e essas informações têm como objetivo principal ampliar a capacidade de tomada de decisão em todos os módulos da arquitetura.

No cenário apresentado na Figura 2.4, o ciclo MAPE-K autonômico é visto como um elemento autonômico, que é dividido em dois grandes blocos: elemento gerenciável e gerenciador autonômico. O elemento gerenciável é o recurso computacional que será controlado pelo gerenciador autonômico. O gerenciador autonômico é composto pelos módulos: monitor, análise, planejamento, execução e conhecimento (explicados anteriormente). O gerenciador autonômico controla o recurso computacional baseando-se em políticas pré-estabelecidas pelo administrador do sistema (Nami e Bertels

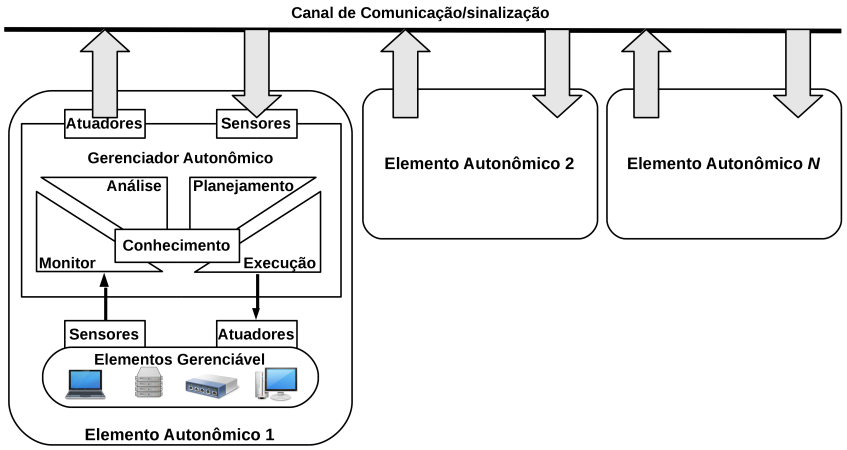


Figura 2.5: *Arquitetura de comunicação entre elementos autonômicos (Nami e Bertels, 2007)*

, 2007). Os sensores e atuadores podem ser vistos dentro ou fora do bloco do gerenciador autonômico. De qualquer forma, devem ser adaptados com interfaces de comunicação entre o elemento e o gerenciador. O ciclo de auto-gerenciamento pode ser imaginado da seguinte forma: os sensores enviam dados ao módulo monitor, esse seleciona informações que merecem ser melhor examinadas pelo módulo de análise. Se alguma condição de risco for identificada é acionado o módulo de planejamento, que escolherá qual ação deve ser tomada para tentar retomar o equilíbrio do sistema. O plano de ação escolhido é repassado ao módulo de execução que dispara ações a serem executadas no elemento gerenciável através dos atuadores, para que o recurso computacional volte a funcionar de acordo com as políticas estipuladas pelo administrador do sistema. O que fecha o ciclo de auto-gerenciamento.

O gerenciador autonômico pode prover ao elemento gerenciável uma ou mais das propriedades auto-X (IBM, 2003).

No nível autonômico global, que permite interação entre elementos autonômicos (ver seção 2.2.2), existe um canal de comunicação que interliga os elementos autonômicos. O objetivo do canal de comunicação é permitir que os elementos autonômicos troquem informações e consigam criar esquemas de gerenciamento autonômico em nível global, fornecendo auto-conhecimento e auto-gerenciamento em nível de negócio. A Figura 2.5 apresenta uma arquitetura de comunicação/ligação entre vários elementos autonômicos. Quando existe interligação entre elementos autonômicos, o gerenciador autonômico ajuda na troca de informações através dos sensores e atuadores. A comunicação entre múltiplos elementos autonômicos é normalmente vista como um barramento simples e a ligação pode ser ponto-a-ponto ou hierárquica (IBM, 2003; Nami e Bertels, 2007).

IBM (2006) apresenta uma arquitetura mais complexa que interliga elementos que compõem os sistemas autonômicos através de camadas, tal arquitetura pode ser vista na Figura 2.6. As camadas apresentadas na Figura 2.6 são formadas por:

1. **Recursos gerenciáveis:** nessa camada estão os elementos gerenciáveis, ou seja, os recursos computacionais a serem gerenciados via CA;

2. **Ponto de interação:** nessa camada estão os sensores e atuadores que devem fornecem uma interface padrão para acessar e gerenciar os elementos gerenciáveis;

3. **Ponto de interação com os gerenciadores autonômicos:** aqui estão os gerenciadores autonômicos que implementam os ciclos autonômicos que permitirão automatizar tarefas administrativas nos elementos gerenciáveis, fornecendo auto-configuração, auto-otimização, auto-cura e auto-proteção. Os gerentes interagem com os elementos gerenciáveis através da camada anterior. Um gerente de ponto de interação pode gerenciar desde um único elemento até grupos de elementos (IBM, 2003). Os gerentes também possuem sensores e atuadores

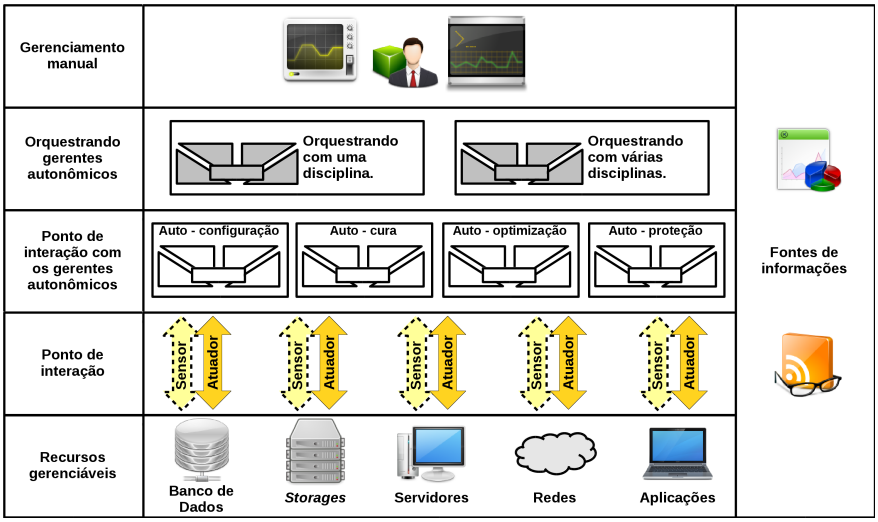


Figura 2.6: *Arquitetura interligando elementos autonômicos em várias camadas (IBM, 2006)*

que fornecem interfaces de acesso à camada superior. As três primeiras camadas seguem os preceitos apresentados anteriormente na Figura 2.4;

4. **Orquestração de gerentes autonômicos:** camada que integra vários gerenciadores autonômicos, permitindo que trabalhem em conjunto. Um gerenciador autonômico normalmente age de forma isolada, gerenciando um único recurso, mas com essa camada é possível coordenar vários gerenciadores autonômicos para que trabalhem em prol de objetivos gerais do sistema, o que torna a arquitetura mais inteligente, pois todo o sistema trabalha em sintonia;

5. **Gerenciamento manual:** fornece interface unificada pela qual o administrador do sistema pode visualizar e interagir com a arquitetura formada pelas camadas anteriores. Normalmente essa interface será utilizada para o acompanhamento de processos, especificação de requisitos, objetivos ou políticas do sistema e em casos mais drásticos para resolver problemas conflitantes não solucionados pela CA.

2.3 Redes definidas por software

O conceito de Redes Definidas por Software - RDS (do inglês, *Software Defined Networking* - SDN) surgiu da necessidade de flexibilizar a forma de controlar/gerenciar as redes de computadores (Lara *et al.*, 2014; Nunes *et al.*, 2014; Paradis, 2014).

Grande parte das tecnologias que são o alicerce para o funcionamento das redes de computadores foram desenvolvidas nas décadas de 1970 e 1980 (Tanenbaum e Wetherall, 2010). São exemplos de tecnologias dessa época o padrão Ethernet, que é muito utilizado em redes locais, e os protocolos IP e TCP, que são a base da Internet. Quando essas tecnologias foram inventadas as redes eram compostas por poucos usuários que acessavam recursos como impressoras e informações em arquivos texto. Porém, as redes de computadores modernas/atuais mudaram significativamente. Por exemplo, agora os usuários podem acessar as redes através de dispositivos móveis e os serviços fornecidos pela rede podem ser textos, hipertextos, áudio e vídeo. Atualmente, também é possível ver o uso em massa de novas mídias, como as redes sociais. Contudo os protocolos que dão suporte a esse novo tipo de acesso a rede continuam sendo os mesmos de décadas passadas.

É bem provável que existam atualmente tecnologias de redes mais modernas que poderiam substituir as antigas e atender de forma bem mais apropriada os requisitos das redes atuais. Porém, sempre surgem receios quando se cogita substituir tecnologias de rede antigas por novas, tal como:

- Os protocolos de rede antigos, mesmo que limitados, são tecnologias consolidadas, ou seja, muitas máquinas trabalham com esses protocolos e talvez uma mudança possa exigir troca de hardware, o que seria dispendioso;
- Também existem preocupações em substituir uma tecnologia já conhecida por outra desconhecida. Neste contexto, os protocolos antigos podem ter problemas, mas esses são conhecidos. Já novas tecnologias podem apresentar problemas ainda desconhecidos.

Um exemplo mais prático é o caso do IPv6 (IP versão 6), que mesmo resolvendo o problema da falta de endereços IPv4 (IP versão 4) válidos na Internet, a anos não consegue se tornar o protocolo padrão da Internet. Desta forma lá se vão décadas criando adendos para que protocolos e tecnologias legadas atendam as exigências atuais de rede (Tanenbaum e Wetherall, 2013).

Outro aspecto comum às redes de computadores é que tanto o software de rede - protocolos e algoritmos que ajudam a manter e gerenciar as redes - quanto as configurações necessárias para o funcionamento da rede ficam aprisionados/embarcados dentro dos hardwares da rede, principalmente *switches* e roteadores. Isso dificulta configurações dinâmicas no ambiente de rede e experimentos com tecnologias emergentes, principalmente novos protocolos das camadas de enlace, rede e transporte. Um problema é que na maioria dos casos os hardwares de rede são proprietários, ou seja fechados, e isso torna difícil a atualização e/ou alteração dos softwares de redes embarcados nesses dispositivos (McKeown *et al.*, 2008). Todavia, ambientes de redes modernos podem exigir alterações constantes em suas configurações para, por exemplo, atender as demandas de QoS ou até mesmo para combater problemas de segurança que estejam afetando a rede.

Da mesma forma, as alterações em configurações ou políticas da rede podem se tornar uma tarefa árdua, já que as redes normalmente são compostas por vários equipamentos de diferentes modelos e fabricantes, que possuem interfaces de configuração/interação distintas, ou seja, não há uma padronização na forma de acessar e controlar esses equipamentos. Assim, é necessário um administrador capacitado e competente para alterar as configurações de uma rede composta por *switches* e roteadores heterogêneos, pois seria preciso executar as alterações equipamento por equipamento, usando possivelmente comandos específicos para cada roteador/*switch*. Essa falta de padronização também torna muito complexa a tarefa de criar softwares capazes de alterar toda a configuração da rede em todos os equipamentos de forma dinâmica, para assim atender as necessidades dos usuários das redes modernas.

Aos problemas supracitados deu-se o nome de "ossificação" da rede/Internet (Lara *et al.*, 2014; McKeown *et al.*, 2008; Nunes *et al.*, 2014), que de forma resumida é a dificuldade em se experimentar novos tipos de protocolos e configurações de redes devido ao fato dos equipamentos de redes não flexibilizarem a forma de controlar/acessar os algoritmos de rede. Uma possível resolução para a ossificação da rede é o uso de Redes Definidas por Software, também chamada de redes programáveis. O conceito de RDS consiste em separar o plano de dados do plano de controle dos equipamentos de rede (McKeown *et al.*, 2008; Nunes *et al.*, 2014; Paradis, 2014).

O plano de dados é a capacidade dos equipamentos de redes de encaminhar pacotes pela rede, ou seja, a função básica de pegar as informações e repassá-las a frente, de um equipamento de rede para outro. Já o plano de controle são as regras que decidem como o encaminhamento de pacotes deve ser feito pelo plano de dados, ou seja, é a parte que dá inteligência ao processo de encaminhamento de pacotes decidindo, por exemplo, qual caminho tomar na rede para entregar informações. Em equipamentos de rede que não utilizam RDS os planos de dados e controle estão entrelaçados e aprisionados dentro do equipamento, o que motiva a ossificação da rede. Então, a solução proposta via RDS é justamente ter os planos de dados e controle bem separados. Assim, o plano de dados ainda fica dentro dos equipamentos da rede. Com o plano de controle totalmente desacoplado, seria permitido alocá-lo, ainda dentro de *switches* e roteadores, ou instalá-lo em computadores dedicados a essa tarefa. Então, com esse novo conceito de rede torna-se possível programar o plano de controle, para que ele se adeque às necessidades de cada rede, daí o nome rede programável. Por fim, o termo Redes Definidas por Software seria então a ideia de retirar o plano de controle do hardware de rede e colocá-lo em nível de software, com o intuito de permitir a reconfiguração ou testes com novos protocolos de rede mais fáceis e factíveis.

Segundo Nunes *et al.* (2014) o conceito de RDS tem atraído a atenção da indústria e da academia, pois representa uma provável evolução para as redes de computadores. Atualmente nota-se alguns esforços para tornar o uso de RDS mais real, tal como a criação de órgãos de padronização e regulamentação para o uso dessa tecnologia. Um exemplo é a ONF (*Open Networking Foundation*) que é patrocinada por empresas como: Google, Yahoo, Facebook, Microsoft, Cisco, Oracle, HP, IBM, Samsung, Vodafone, Telefônica, VmWare, dentre inúmeras outras (ONF, 2014).

A primeira proposta que apresentou alguns aspectos de RDS na prática foi a SOFTNET (Zander e Forchheimer, 1988), que utilizava os pacotes de rede para enviar comandos que alteravam a forma de operação dos equipamentos de uma rede de rádio comutada por pacotes. Inspirado no SOFTNET surgiu o AN (*Active Networks*) (Smith e Nettles, 2004) que traz a ideia de elementos de rede que podem executar programas que são enviados através dos pacotes da rede. Posteriormente foi proposta uma nova versão do AN chamada de NetServ (Lee *et al.*, 2011). A principal característica do SOFTNET e do AN é que esses não usam componentes de software para controlar a rede, mas sim o envio de comandos embutidos na carga útil dos pacotes de rede. Portanto, nessas propostas não há um componente de software sendo executado em um servidor ou CPU dedicada, o que segundo Lara *et al.* (2014) não é considerado RDS, pois não desacopla totalmente o plano de controle do plano de dados, mas tal assunto ainda gera discussão.

Conforme Lara *et al.* (2014); Nunes *et al.* (2014) existem atualmente duas propostas mais concretas de RDS, que são: ForCES (*Forwarding and Control Element Separation*) e OpenFlow. O ForCES (Doria *et al.*, 2010) redefine a arquitetura interna dos dispositivos de rede, separando o plano de controle do plano de dados, contudo na abordagem proposta pelo ForCES, ambos planos continuam dentro de um mesmo dispositivo, ou seja, não há a visão de um dispositivo independente para o processamento do plano de controle nessa abordagem. O ForCES é uma criação do IETF (*Internet Engineering Task Force*) para padronizar a forma de comunicação entre os elementos controladores da rede e os elementos controlados. O ForCES é composto por três elementos, que são: elemento de encaminhamento, elemento de controle e protocolo de comunicação. Sendo que o elemento de encaminhamento é responsável por conduzir pacotes pela rede utilizando para isso informações que são repassadas pelo elemento de controle. O protocolo de comunicação ForCES é utilizado para comunicação e sincronização entre os elementos de controle e de encaminhamento. Os elementos da rede ForCES trabalham utilizando um esquema mestre/escravo. Tal como o ForCES, existem outras propostas de implementação de RDS (Lara *et al.*, 2014; Nunes *et al.*, 2014), contudo nenhuma delas foi tão bem aceita pela indústria e academia como o OpenFlow, que será apresentado a seguir e serve de base para o presente trabalho.

2.3.1 OpenFlow

O OpenFlow foi idealizado com o objetivo de tratar o problema da ossificação, que dificultava a experimentação de novas tecnologias de rede e configurações dinâmicas, principalmente em ambientes heterogêneos de rede. No trabalho (McKeown *et al.*, 2008) os autores descrevem seus anseios em poder executar protocolos experimentais em ambientes de produção reais, sem afetar outros usuários, bem como da necessidade de configurar dinamicamente e através de uma interface única os equipamentos de rede. Assim, McKeown *et al.* (2008) apresentam o OpenFlow que fornece um padrão de interface de comunicação aberta que permite separar o plano de controle do plano de dados. Dessa forma, o OpenFlow coloca o plano de controle, que é responsável por comandar a rede em nível de software, o que permite a programação das redes e torna palpável a ideologia de RDS.

Uma rede OpenFlow é composta basicamente por três elementos, que são:

- ***Switch* OpenFlow**, que é o elemento que mantém o plano de dados e fica responsável pelo encaminhamento dos pacotes pela rede;
- **Controlador OpenFlow**, que é o dispositivo responsável pelo plano de controle da rede, ou seja, é esse dispositivo que permite controlar a rede e programá-la de acordo com as necessidades dos usuários;

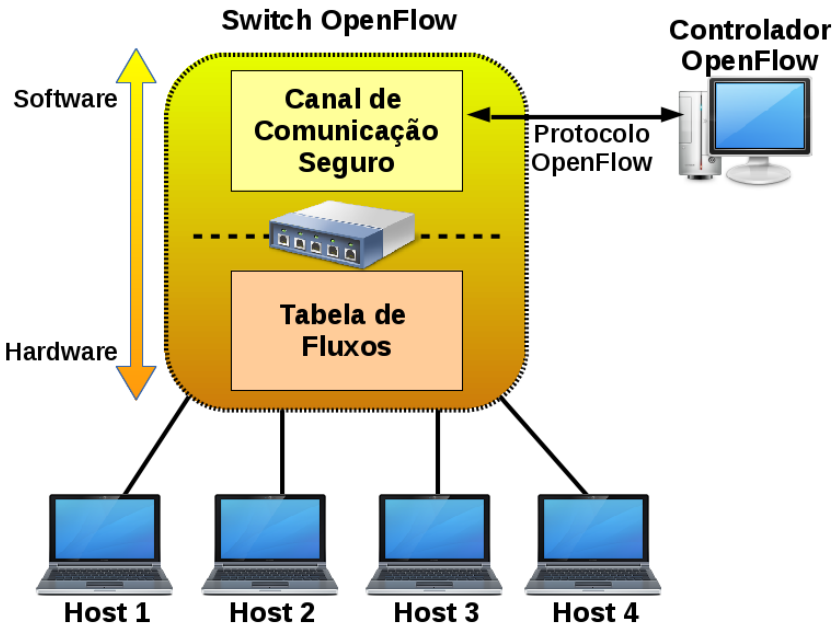


Figura 2.7: *Elementos de uma rede OpenFlow (McKeown et al., 2008)*

- **Protocolo OpenFlow** que é responsável pela intercomunicação entre o *switch* e o controlador OpenFlow.

A Figura 2.7 ilustra os elementos da rede OpenFlow, que de forma simplista é composta por *hosts* e *switch*. A única e grande alteração é que um desses *hosts* será o controlador OpenFlow.

A primeira vista uma rede utilizando OpenFlow parece uma rede Ethernet comum, a diferença é que os *switches* ou roteadores OpenFlow (McKeown *et al.*, 2008) possuem tabelas de fluxos (ver Figura 2.7) para encaminhar os pacotes. Mais especificamente, as tabelas de fluxos são regras/informações sobre como encaminhar um pacote pela rede, por exemplo, uma entrada na tabela de fluxo pode ser alguma informação como:

- "*Pacotes originados do host 1 entrando pela porta 1 do switch e destinados ao host 4, devem ser entregues através da porta 4 do switch* ".

Só que *switches* OpenFlow não ficam restritos às informações da camada de enlace, como os *switches* comuns, pois podem utilizar informações de outras camadas e desta forma é possível ter regras mais complexas na tabela de fluxos, tal como:

- "*Pacotes originados do host 2, com o IP 10.0.0.2, usando o protocolo TCP, com porta de origem 51000, entrando pela porta 1 do switch e destinados ao host 4, com IP 10.0.0.4, porta 80, devem ser entregues através da porta 4 do switch* "

Assim, uma tabela de fluxo é uma lista na qual cada linha representa um fluxo de rede. Quando um pacote chega ao *switch* esse é comparado com cada linha dessa lista. Caso exista um fluxo que combine com o pacote, então esse pacote é enviado pela rede utilizando as informações desse fluxo para alcançar o destino. Caso não exista uma entrada na tabela de fluxos que combine com o pacote, esse pacote pode ser bloqueado; ou ser enviado do *switch* para o controlador OpenFlow (OpenFlow, 2011). O controlador, por sua vez, deverá analisar as informações desse pacote e decidir como encaminhá-lo pela rede. A decisão de como enviar os pacotes pela rede é programável no controlador em nível de software, daí a relação do OpenFlow com o conceito de RDS. Uma vez que o controlador tenha decidido qual ação deve ser tomada para que o pacote chegue ao destino, o controlador adiciona, via protocolo OpenFlow, uma nova entrada para esse pacote na tabela de fluxo do *switch*. A partir de então os pacotes subsequentes que combinem com esse fluxo, instalado no *switch*, irão seguir pela rede observando apenas as informações do fluxo instalado no *switch* e

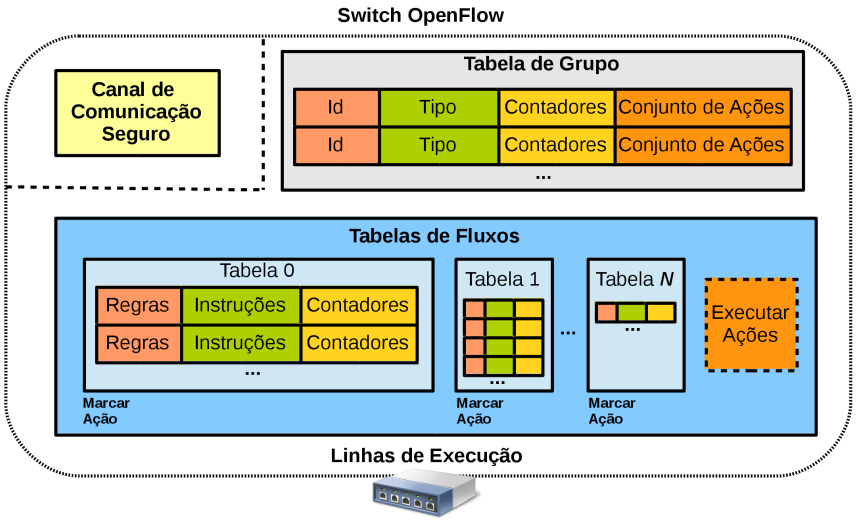


Figura 2.8: *Switch OpenFlow (Nunes et al., 2014; OpenFlow, 2011)*

portanto não precisam ser enviados para o controlador. Em resumo, em uma rede OpenFlow existem dois casos para o tratamento de pacotes:

1. Quando o pacote não possui uma entrada no *switch*. Nesse caso o pacote é tratado tanto pelo *switch* quanto pelo controlador;

2. Quando o pacote já possui uma entrada na tabela de fluxos do *switch*. Nesse caso o pacote só é tratado pelo *switch*, não sendo necessário envia-lo para o controlador.

Os *switches* OpenFlow devem descartar entradas para fluxos ociosos, ou seja, se uma entrada para um fluxo não receber pacotes durante um breve período de tempo esse fluxo deve ser removido.

As tabelas de fluxos dos *switches* inicialmente estão sem nenhuma entrada, pois as tabelas de fluxos dos *switches* devem ser povoadas pelo controlador OpenFlow. Idealmente, apenas o primeiro pacote de uma transmissão de rede entre duas máquinas é enviado ao controlador e todos os pacotes subsequentes referentes a essa transmissão seguirão passando apenas pelo *switch*. Ao primeiro pacote e seus subsequentes dá se o nome de fluxo de rede, daí o nome da tabela de fluxos e da tecnologia OpenFlow, no qual *flow* significa fluxo e *open* remete à ideia de aberto, livre, liberdade, tal como código fonte aberto - OpenSource[6]. Atualmente existem várias versões do protocolo OpenFlow, sendo a primeira a 1.0 e a última, até a data de redação desse texto, a 1.4 (ONF, 2013). As seções seguintes descrevem mais detalhadamente as partes da arquitetura OpenFlow e sua interação.

2.3.1.1 *Switch*

Na tecnologia OpenFlow os *switches* são responsáveis pelo encaminhamento dos pacotes pela rede, só que gerenciados pelos controladores OpenFlow. A arquitetura do *switch* OpenFlow é apresentada na Figura 2.8.

Um dos principais elementos dentro de um *switch* OpenFlow são as tabelas de fluxos. As tabelas de fluxos são listas com entradas que indicam o que fazer com os pacotes que estão chegando ao *switch*. Até a versão 1.2 do protocolo OpenFlow (ONF, 2011) as entradas das tabelas de fluxos eram formadas apenas pelos campos regras, instruções e contadores, como na Figura 2.8. A partir da versão 1.3 do OpenFlow (ONF, 2012) foram adicionados mais campos (ver Figura 2.9), sendo a função desses campos:

- Regras (*match fields*): são informações que poderão ser utilizadas para comparar se um pacote entrando no *switch* combina ou não com a entrada da tabela de fluxos (Nunes *et al.*, 2014), esses dados são:

[6] Maiores informações estão disponíveis em http://opensource.org/.

Figura 2.9: *Campos que constituem as entradas nas tabelas de fluxo a partir da versão 1.3 do OpenFlow (ONF, 2012)*

  - Porta de entrada no *switch*;
  - Endereço Ethernet de origem/destino;
  - Informações a respeito de VLAN (*Virtual LAN*);
  - Informações a respeito de MPLS (*Multiprotocol Label Switching*);
  - Informações sobre protocolos (ARP (*Address Resolution Protocol*), IP, ICMP, UDP, SCTP (*Stream Control Transmission Protocol*) (IETF, 2002) e TCP);
  - Endereço IP de origem/destino;
  - Portas que identifiquem serviços da camada de aplicação;
  - Metadados (informações que são repassadas entre as tabelas de fluxo).

- Instruções: são utilizadas para modificar o conjunto de ações, campos dos cabeçalhos e linha de execução do pacote;

- Contadores: são estatísticas a respeito do uso da entrada em questão, portanto podem ser informações como *bytes* enviados/recebidos, número de pacotes tratados, tempo de duração da entrada, etc.

- Prioridade: indica a precedência de correspondência para uma entrada que combina com o pacote. Uma entrada na tabela de fluxo é identificada unicamente pelos campos de regras e prioridade;

- Limite de tempo (*timeouts*): determina o tempo que o fluxo/entrada será mantido no *switch*. Um fluxo pode ser removido automaticamente pelo *switch* caso fique ocioso ou também é possível determinar um período de tempo fixo que o fluxo permanecerá no *switch*;

- *Cookie*: Pode ser usado pelo controlador para filtrar estatísticas de fluxos, modificações nos fluxos ou para apagar fluxos. Não é usado no processamento.

Quando o pacote chega no *switch* deve ser selecionada a entrada, na tabela de fluxos, com prioridade mais alta e que combina com os valores do campo de regras. Caso haja entradas conflitantes quanto à prioridade e campos das regras, nenhuma dessas entradas será escolhida.

Os *switches* OpenFlow podem ter várias linhas de processamento de pacotes (OpenFlow *pipeline*). Um *switch* híbrido, tem uma linha de processamento OpenFlow e outra que representa a comutação normal Ethernet, cabe ao administrador programar quando usar a linha de processamento normal ou a OpenFlow. Em *switches* somente OpenFlow (não híbridos) a linha de processamento define como os pacotes interagem com as várias tabelas de fluxos que o *switch* pode ter, mas também é possível trabalhar com uma única tabela de fluxo.

Em *switches* OpenFlow os pacotes sempre iniciam o processamento pela primeira tabela do *switch*, que é a tabela 0 (zero). Se o pacote combinar com as regras de uma entrada, em uma dada tabela, as instruções referentes a essa entrada são executadas. As instruções podem ser desde aplicar uma sequência de ações ou até mesmo enviar o pacote para ser processado por outra tabela de fluxos (Flowgrammable, 2014)[7]. Porém, o pacote só pode ir para frente nas tabelas, ou seja, um pacote não pode ir da tabela 1 para a tabela 0. Se o pacote não combinar com nenhuma das entradas das tabelas de fluxo presentes no *switch* OpenFlow, ele deve ser submetido a uma entrada para pacotes

[7]Sítio Web, com conteúdo bem organizado e dedicado ao OpenFlow e RDS, mantido por Jasson Casy, que é associado ao ONF e candidato a PhD na Texas A&M *University*. Fazem parte da equipe do sítio também outros profissionais ligados a indústria e academia.

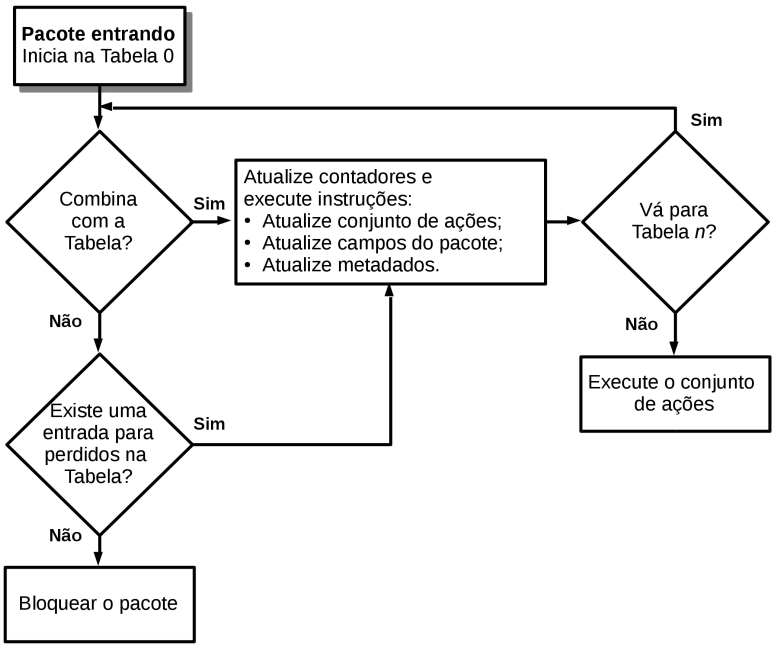


Figura 2.10: *Fluxograma dos pacotes nas tabelas de fluxos (ONF, 2013)*

perdidos na tabela fluxos (*table-miss flow entry*), que dependendo da configuração, pode executar as seguintes ações: bloquear; enviar para outra tabela; ou enviar o pacote para o controlador (ONF , 2013). A entrada para pacotes perdidos é simbolizada por uma entrada com a menor prioridade possível, nesse caso 0 (zero), e com todos os valores dos campos de regras omitidos (endereços, protocolos, etc). Todas as tabelas de fluxo podem possuir uma entrada de pacotes perdidos. Se a entrada para pacotes perdidos não existir, o comportamento padrão é descartar/bloquear os pacotes que não combinarem com as entradas nas tabelas de fluxos. A Figura 2.10 apresenta os possíveis caminhos que um pacote pode enfrentar em um *switch* OpenFlow.

O encaminhamento de pacotes, feito pelos *switches* OpenFlow, pode ser realizado por portas físicas ou virtuais. As portas físicas são as portas reais, implementadas no hardware do *switch*, já as portas virtuais podem especificar ações genéricas como: enviar o pacote para o controlador; encaminhar o pacote para todas as portas físicas do *switch* (*flooding*); ou enviar o pacote usando um método não OpenFlow, tal como, processar o pacote como se fosse um *switch* comum. A versão 1.4 do OpenFlow (ONF, 2013) especifica além das portas físicas e virtuais as portas reservadas, que são justamente a porta usada para comunicação com o controlador e as portas não-OpenFlow. As entradas das tabelas de fluxo também podem apontar para tabelas de grupos OpenFlow, que especificam se algum processamento adicional deve ser feito, tal como, encaminhar para outras portas do *switch* (OpenFlow, 2011).

Uma tabela de grupo especifica se há necessidade de processamento adicional sobre o pacote analisado pelo *switch* OpenFlow. Os grupos representam conjuntos de ações, tal como, encaminhamentos para múltiplas portas. Conforme Figura 2.8, a tabela de grupos é composta por entradas, que são formadas pelos campos:

- Id (Número de Identificação) do grupo, que é um número único que serve para identificar o grupo;
- Tipo, identifica a semântica do grupo, por exemplo, o tipo *all* executa todos os conjuntos de ações em um grupo, o tipo *select* executa um conjunto de ações em um grupo (OpenFlow, 2011);
- Contadores, que são atualizados quando pacotes são processados pelo grupo;

- Conjunto de ações do grupo (*action buckets*), que é uma lista ordenada de conjuntos de ações a serem executadas.

Os grupos possibilitam que múltiplos fluxos sejam encaminhados utilizando um único identificador. Isso permite criar métodos adicionais de encaminhamento e que ações de encaminhamento possam ser alteradas facilmente.

As instruções das tabelas de fluxos não existiam antes da versão 1.1.0 do OpenFlow, ou seja, na versão 1.0 as ações eram aplicadas diretamente pelas lista de ações usando mensagens OpenFlow (*FlowMod*). Essas ações podiam ser: descartar, modificar, enfileirar ou encaminhar o pacote. Portando, desde a versão 1.1.0 os pacotes que combinam com as regras de uma entrada das tabelas de fluxos devem ser submetidos às instruções (OpenFlow, 2011), que por sua vez levam à execução de ações nos pacotes. As instruções podem ser utilizadas para modificar pacotes e/ou encaminhá-los para portas, tabelas OpenFlow, ou outras linhas de processamento. Tais instruções normalmente são:

- Aplicar (*apply*): Aplica a ação ou ações imediatamente no pacote;
- Limpar (*clear*): Limpa o conjunto de ações;
- Gravar ações (*write action*): Grava/adiciona ações no conjunto de ações atual;
- Gravar metadados (*write metadata*): Gravar valores no campo de metadados;
- Ir para tabela (*goto-table*): Continua o processamento do pacote em outra tabela que é especificada na instrução;
- Experimentar (*experimenter*): Instrução para experimentos;

Quando um pacote adentra ao *switch* ações podem ser aplicadas/relacionadas a esse pacote, para isso o pacote é vinculado a um conjunto de ações. No início esse conjunto de ações está vazio, mas à medida que o pacote for combinando com as entradas das tabelas de fluxo as instruções de *Gravar* ou *Aplicar* irão respectivamente adicionando ou aplicando ações ao pacote. Quando não há mais entradas nas tabelas de fluxo que combinam com o pacote o seu processamento chega ao fim e então são executadas as ações que estão no conjunto de ações do pacote. Também há a possibilidade do pacote não combinar com nenhuma entrada das tabelas de fluxo, o que deixaria o conjunto ações vazio (sem nenhuma ação), então esse pacote seria submetido a entrada para pacotes perdidos da tabela de fluxos e uma das ações poderia ser enviá-lo para o controlador OpenFlow (ver Figura 2.10).

A interação entre as instruções e o processamento do pacote é ilustrado na Figura 2.11, sendo possível observar que as ações de *Gravar* e *Limpar* são responsáveis por manipular o conjunto de ações do pacote. A instrução *Aplicar* altera diretamente informações do pacote e as ações de *Gravar ação* e *Ir para tabela* são utilizadas para alterar informações dos metadados do pacote. As instruções *Aplicar* e *Gravar* são as únicas que realmente executam ações no pacote e ambas têm como entrada uma lista de ações a serem aplicadas imediatamente ao pacote. A diferença entre as instruções *Gravar* e *Aplicar* é que na primeira as ações selecionadas para o pacote só são executadas no final do processamento de todas as entradas das tabelas de fluxo, ou seja, as ações vão sendo armazenadas conforme forem combinando com as entradas das tabelas de fluxos e são executadas todas no final do processamento das tabelas de fluxo. Já na segunda (*Aplicar*), as ações são executadas imediatamente, sem a necessidade de ficarem armazenadas no conjunto de ações e sem precisar esperar pelo final do processamento do pacote com as entradas das tabelas de fluxo. A instrução *Experimentar* permite que pesquisadores façam suas próprias instruções OpenFlow.

A partir da versão 1.3.0 do OpenFlow também foi adicionada a instrução *Meter* que permite regular a taxa de transferência de dados (Flowgrammable, 2014; ONF, 2012). A instrução *Meter* submete o pacote a uma nova tabela chamada também de *Meter*, que permite implementar QoS

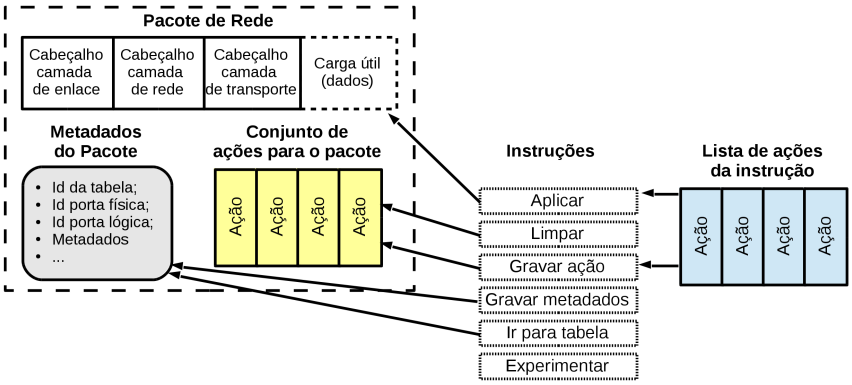


Figura 2.11: *Interação entre as instruções OpenFlow e o pacote de rede (Flowgrammable, 2014)*

em redes OpenFlow. Antes da tabela *Meter* só era permitido limitar a largura de banda[8] por porta, usando por exemplo a ação fila (*queue*) (apresentada posteriormente), mas com o *Meter* é possível criar controles personalizados para cada fluxo e não apenas por portas. Uma entrada na tabela *Meter* é composta pelos campos *Meter* Id, que identifica a entrada; *Meter band*, uma lista que especifica a largura de banda ou a forma de processamento que será aplicado ao pacote; e contadores, que são atualizados quando a entrada é utilizada. Cada entrada na lista *Meter band* é formada basicamente por informações como o tipo de processamento (*band type*)[9], taxa de transferência a ser aplicada (*band*) e contadores. Por fim, em alguns casos um dado *switch* pode não dar suporte a uma determinada instrução, nesse caso o *switch* deve retornar uma mensagem OpenFlow de erro.

Um dos grandes diferenciais da tecnologia OpenFlow é a possibilidade de executar diversas ações nos pacotes que cruzam a rede e não só a ação de encaminhamento de pacotes. A Tabela 2.1 apresenta as possíveis ações que são aplicáveis aos pacotes processados por *switches* OpenFlow, bem como relaciona tais ações com as situações/protocolos em que são aplicáveis. Dessa forma, observando a Tabela 2.1 é nota-se que as possíveis ações OpenFlow são:

- Bloquear: usado para descartar pacotes;
- Encaminhar: envia pacotes para portas de saída, tabelas de fluxo ou tabelas de grupos;
- Empilhar/desempilhar: utilizado para empilhar e desempilhar informações como, Id de VLAN ou MPLS;
- Alterar: troca valores dos campos dos cabeçalhos do pacote de rede;
- Decrementar: parecido com a ação *Alterar*, só que utilizado para decrementar campos dos cabeçalhos, como o tempo de vida dos pacotes na rede - TTL (*Time to Live*) (Tanenbaum e Wetherall , 2010);
- Copiar entrada/saída: Também altera campos dos cabeçalhos do pacote, só que copiando dados de um cabeçalho para outro. Exemplos seriam as cópias de campos entre os protocolos: MPLS para MPLS; MPLS para IP; IP para MPLS, dentre outras combinações.

Um *switch* não obrigatoriamente irá suportar todas as ações, então ele deve anunciar suas capacidades para o controlador OpenFlow.

[8] nesse texto considera-se largura de banda como sendo sinônimo de taxa de transferência de dados e não com o significado de propriedades ou limitações físicas do meio de transmissão (Tanenbaum e Wetherall, 2010).

[9] as especificações OpenFlow ONF (2012, 2013) não descrevem os tipos de processamento aplicáveis aos pacotes.

Tabela 2.1: *Tipos de Ações (Flowgrammable, 2014; OpenFlow, 2011)*

Protocolo	Ação	Objetivo ou campo do cabeçalho do protocolo em que se aplica
-	Bloquear	Descartar pacotes.
Saída	Encaminhar	Enviar o pacote para uma única porta de saída (*Port* Id).
		Enviar o pacote para todas portas do *switch* (ALL).
		Enviar o pacote para a porta que ele ingressou (IN_PORT/INGRESS).
		Enviar o pacote para o controlador (CONTROLLER).
		Envia o pacote para a tabela de fluxos 0 (TABLE).
		Permite usar portas comuns do *switch* para enviar pacotes OpenFlow (LOCAL).
		*Enviar pacotes para todas as portas não OpenFlow do *switch*, usando o método tradicional (FLOOD).
		*Enviar para a linha de processamento de pacotes não OpenFlow- tradicional (NORMAL).
Grupo	Encaminhar	Aplicar um processamento de grupo através do número de identificação do grupo (*Group* Id).
*Fila	Alterar	Alterar o Id da fila
*Ethernet	Alterar	Endereço de origem.
		Endereço de destino.
		VLAN Id.
		Prioridade de VLAN.
		PBB (*Provider Backbone Bridging*) I-SID (*Instance Service Identifier*) (IEEE, 2008).
	Empilhar/desempilhar	VLAN Id.
		PBB I-SID.
*MPLS	Alterar	Rótulo (*label*).
		Classe do tráfego.
		TTL.
		Final da fila.
	Decrementar	TTL.
	Empilhar/desempilhar	Rótulo (*label*).
	Copiar entrada	TTL: copiar o TTL do cabeçalho externo para o próximo cabeçalho mais externo. Exemplo: MPLS-para-MPLS, MPLS-para-IP e IP-para-IP.
	Copiar saída	TTL: copiar o TTL do cabeçalho interno para o próximo cabeçalho mais interno.
*ARP	Alterar	Código de Operação.
		Endereço de hardware da origem.
		Endereço de protocolo da origem.
		Endereço de hardware do destino.
		Endereço de protocolo do destino.
*IPv4	Alterar	Endereço de origem.
		Endereço de destino.
		ToS (*Type of Service*).
		ECN (*Explicit Congestion Notification*).
		TTL.
	Decrementar	TTL.
	Copiar entrada	TTL: copiar o TTL do cabeçalho externo para o próximo cabeçalho mais externo.
	Copiar saída	TTL: copiar o TTL do cabeçalho interno para o próximo cabeçalho mais interno.
*IPv6	Alterar	Endereço de origem.
		Endereço de destino.
		ToS.
		ECN.
		Rótulo de fluxo (*flow label*).
		TTL.
		Cabeçalho de extensão (*extension header*).
	Decrementar	TTL.
	Copiar entrada	TTL: copiar o TTL do cabeçalho externo para o próximo cabeçalho mais externo.
	Copiar saída	TTL: copiar o TTL do cabeçalho interno para o próximo cabeçalho mais interno.
*TCP	Alterar	Porta de origem.
		Porta de destino.
*UDP	Alterar	Porta de origem.
		Porta de destino.
*SCTP	Alterar	Porta de origem.
		Porta de destino.
*ICMPv4	Alterar	Tipo (*type*).
		Código (*code*).
*ICMPv6	Alterar	Tipo (*type*).
		Código (*code*).
		ND (*Neighbor Discovery*) - endereço do destino.
		ND - endereço de origem na camada de enlace.
		ND - endereço de destino na camada de enlace.
Experimental	-	-

***** - Opcional ao *switch*.

Conforme a Tabela 2.1, várias ações podem ser aplicadas em um mesmo protocolo/contexto, ou seja, podem ser aplicadas em tabelas de grupos (*group*), portas de saída (*output*), filas (*queue*) e nos cabeçalhos dos protocolos encapsulados no pacote analisado.

Uma ação muito importante e obrigatória é a de encaminhar para portas de saída (*output*). Portas de saída podem ser físicas ou virtuais. Os *switches* utilizam alguns nomes de portas que já possuem funções pré-definidas, alguns nomes são (OpenFlow, 2011):

- ALL: representa todas as portas excluindo a porta pela qual o pacote entrou no *switch* ou portas configuradas com a opção OFPPC_NO_FWD. Normalmente é utilizada para difundir informações por toda a rede;
- CONTROLLER: representa a porta que conecta o controlador OpenFlow. Usada para enviar pacotes para o controlador;
- TABLE: usada para submeter o pacote à primeira tabela de fluxos;
- IN_PORT: representa a porta pela qual o pacote entrou no *switch*;
- LOCAL: usada para permitir que controladores remotos acessem o *switch* via protocolo OpenFlow utilizando as portas comuns do *switch* (não a CONTROLLER). O *switch* é normalmente interligado ao controlador OpenFlow através de uma rede isolada, a porta LOCAL permite que o controlador acesse o *switch* via protocolo OpenFlow utilizando a rede normal na qual estão conectados todos os *hosts*, assim essa porta permite, por exemplo, a intercomunicação entre controladores e *switches* sem a necessidade de uma estrutura de rede isolada para isso;
- NORMAL: processa o pacote usando a linha de execução tradicional (não OpenFlow);
- FLOOD: envia o pacote para todas as portas do *switch* usando a linha de execução tradicional (não OpenFlow)
- ANY: utilizada em alguns comandos OpenFlow para informar que essa porta combina com qualquer valor. É uma espécie de porta curinga.

Os nomes/funções LOCAL, NORMAL e FLOOD são opcionais, desta forma alguns *switches* podem simplesmente não implementar tais funções ou em outros casos podem necessitar de configurações para provê-las.

A ação de encaminhamento para tabela de grupos é implementada por todos os *switches* OpenFlow e permite agregar várias ações aos pacotes através das entradas na tabela de grupos (mencionado anteriormente).

Switches OpenFlow também podem implementar esquemas de QoS. Isso é possível através da criação de filas (*queue*) para regular a taxa de transferência. É possível que uma porta de saída física seja representada por várias portas lógicas através das filas, sendo que cada uma pode usar esquemas de prioridade ou taxa de transferência de dados distintas. Para usar essas filas basta alterar o Id da fila durante o processamento do pacote. Assim por exemplo, um pacote com Id de fila 0 (padrão) sai pela porta usando toda a largura de banda da porta, já um pacote que tenha o seu Id de fila alterado para um valor diferente de 0 pode sofrer restrições ou ganhar privilégios durante sua transmissão pela porta física. A configuração das filas é feita normalmente nas configurações do *switch*, o OpenFlow apenas permite alterar o pacote/fluxo para usar ou não tais filas.

As ações também são comumente aplicadas aos cabeçalhos de protocolos encapsulados no quadro de rede analisado pelo *switch* OpenFlow. Conforme a Tabela 2.1, utilizando OpenFlow pode-se tratar informações de protocolos da camada de enlace (1) até a camada de transporte (4). Desde a versão 1.0 (Flowgrammable, 2014) do OpenFlow já é factível programar campos dos protocolos: Ethernet, IPv4, UDP e TCP. Na versão 1.1 foram incluídos os protocolos: MPLS e SCTP. E a partir da versão 1.2 também são suportados os protocolos: ARP, IPv6, ICMPv4 (ICMP versão 4) e ICMPv6 (ICMP versão 6).

Dentre as ações aplicáveis aos cabeçalhos dos protocolos, a ação *Alterar* (*set*) permite mudar valores de campos dos cabeçalhos, possibilitando, por exemplo, trocar um endereço IP de origem por outro IP, ou mudar as portas de destino de um pacote TCP. A ação *Decrementar* (*decrement*) permite diminuir o contador do campo TTL dos protocolos MPLS e IP. Já as ações *copiar entrada* (*copy inwards/in*) e *copiar saída* (*copy outwards/out*) permite copiar campos de um protocolo para outro, tal como copiar o valor do campo TTL do cabeçalho MPLS para o mesmo campo do cabeçalho IP do quadro de rede que está sendo processado, ou copiar do IP para o MPLS. Também pode-se utilizar as opções *empilhar/desempilhar* (*push/pop*), que provê a capacidade de acrescentar e retirar cabeçalhos no quadro de rede, assim por exemplo, é possível que um dado quadro de rede pertença a várias VLANs. Para isso um *switch* OpenFlow pode acrescentar um cabeçalho de VLAN utilizado a ação *empilhar* e depois, quando necessário, retirá-lo usando a ação *desempilhar*.

Sempre que uma instrução/ação é aplicada a um pacote são atualizados seus contadores. Desta forma, os *switches* OpenFlow possibilitam a obtenção de informações a respeito do estado atual da rede, pois permitem o resgate de dados referentes a tabelas, fluxos, portas, filas e grupos. Esses dados podem ser extraídos dos contadores dos *switches* e armazenados para futuramente serem utilizados como estatísticas da rede. A Tabela 2.2 apresenta os possíveis contadores definidos no OpenFlow (ONF, 2012).

O *switch* OpenFlow, assim como o controlador, possui em sua arquitetura um canal de comunicação que permite a troca de informações entre os elementos da rede OpenFlow, ou seja, entre *switch* e controlador. A seção 2.3.1.2, apresentada a seguir, detalha mais o canal de comunicação, bem como o protocolo OpenFlow.

2.3.1.2 Canal de comunicação e protocolo

O canal de comunicação serve de interface entre o plano de controle e o plano de dados, que são respectivamente representados pelo controlador e *switch*. A troca de informações pelo canal de comunicação se dá via protocolo OpenFlow (Nunes *et al.*, 2014).

O canal de comunicação no *switch* é simbolizado pela porta CONTROLLER e é utilizada para enviar estatísticas a respeito da rede/fluxos e principalmente para enviar pacotes que não possuem fluxos instalados no *switch* que devem ser analisados pelo controlador, para que o plano de controle decida como programar as tabelas do *switch*. Já o controlador utiliza esse canal para enviar comandos para o *switch* e assim gerenciar/programar o *switch*.

Caso haja interrupção no canal de comunicação, o comportamento padrão do *switch* OpenFlow é entrar em um modo de segurança (*fail secure mode*), o que irá bloquear todos os pacotes novos que chegarem no *switch*. *Switches* híbridos também podem disponibilizar o modo independente (*fail standalone mode*) que realiza comutação tradicional caso ocorram falhas de comunicação com o controlador. A partir da versão 1.2 do OpenFlow é possível que *switches* suportem o controle de múltiplos controladores (Lara *et al.*, 2013). Isso pode ajudar a evitar a interrupção da comunicação entre controlador e *switch*.

A comunicação entre o plano de dados e o plano de controle pode ser feita utilizando um canal TCP puro, mas é recomendável utilizar o suporte a TLS (*Transport Layer Security*). Usando TLS, os dados seguem pela rede criptografados, o que dificulta interceptação compreensível e principalmente ajuda a garantir a autenticidade dos comandos, ou seja, assegura que o *host* enviando comandos é realmente o controlador legítimo e não um impostor (ONF, 2012).

Toda interação entre os elementos da rede OpenFlow só é possível graças ao protocolo OpenFlow, que tem por funções: definir a estrutura e semântica das mensagens trocadas entre os elementos OpenFlow; descrever como ocorre a negociação entre versões e capacidades dos elementos; e fornecer uma interface de comunicação com o mundo externo (Flowgrammable, 2014). Conforme ONF (2013) as mensagens podem ser classificadas segundo o tipo de comunicação, que são: assíncronas, simétricas e controlador-para-*switch*. A Tabela 2.3 apresenta as mensagens OpenFlow, uma breve descrição, o tipo de comunicação e a versão OpenFlow que dá suporte a cada mensagem. Nesse cenário, mensagens do tipo controlador-para-*switch*, que na Tabela 2.3 foi abreviada para Crtler⇒*Switch*, são mensagens enviadas/iniciadas pelo controlador para gerenciar ou obter

Tabela 2.2: *Contadores presentes no OpenFlow (ONF, 2012).*

Nome do contador	Descrição
Por Tabela de Fluxo	
**Reference Count*	Número de entradas ativas na tabela.
Packet Lookups	Número de pacotes que foram submetidos a tabela.
Packet Matches	Número de pacotes que combinaram com a tabela.
Por Fluxo	
Received Packets	Número de pacotes recebidos.
Received Bytes	Quantidade de bytes recebidos.
**Duration in Seconds*	Duração em segundos.
Duration in Nanoseconds	Duração em nanosegundos.
Por Porta	
**Received Packets*	Número de pacotes recebidos.
**Transmitted Packets*	Número de pacotes transmitidos.
Received Bytes	Quantidade de bytes recebidos.
Transmitted Bytes	Quantidade de bytes transmitidos.
Receive Drops	Número de pacotes recebidos bloqueados.
Transmit Drops	Número de pacotes transmitidos bloqueados.
Receive Errors	Recebidos com erro.
Transmit Errors	Transmitidos com erro.
Receive Frame Alignment Errors	Número de quadros recebidos com erros de alinhamento.
Receive Overrun Errors	Recebidos com problemas de saturação (capacidade de processamento/memória foi excedida pela capacidade recebimento).
Receive CRC Errors	Recebidos com erro de soma de verificação.
Collisions	Número de colisões.
**Duration in Seconds*	Duração em segundos.
Duration in Nanoseconds	Duração em nanosegundos.
Por Fila	
Transmit Packets	Número de pacotes transmitidos.
Transmit Bytes	Quantidade de bytes transmitidos.
Transmit Overrun Errors	Transmitidos com problema de saturação.
**Duration in Seconds*	Duração em segundos.
Duration in Nanoseconds	Duração em nanosegundos.
Por Grupo	
Reference Count	Número de entradas ativas na tabela.
Packet Count	Número de pacotes que combinaram com o grupo.
Byte Count	Quantidade de bytes tratados.
**Duration in Seconds*	Duração em segundos.
Duration in Nanoseconds	Duração em nanosegundos.
Por Bucket	
Packet Count	Número de pacotes que combinaram com o grupo.
Byte Count	Quantidade de bytes tratados.
Por Meter	
Flow Count	Número de fluxos.
Input Packet Count	Número de pacotes entrando.
Input Byte Count	Quantidade de bytes entrando.
**Duration in Seconds*	Duração em segundos.
Duration in Nanoseconds	Duração em nanosegundos.
Por Meter Band	
In Band Packet Count	Número de pacotes entrando.
In Band Byte Count	Quantidade de bytes entrando.

***** - Obrigatório.

informações do *switch*. Já mensagens assíncronas são enviadas/iniciadas do *switch* e servem para informar ao controlador sobre eventos que ocorrem no *switch*. Por fim, mensagens simétricas são enviadas/iniciadas sem solicitação por qualquer elemento da rede OpenFlow.

Tabela 2.3: *Mensagens OpenFlow (Flowgrammable, 2014)*

Mensagem	Descrição	Comunicação	Versão do Protocolo				
			1.0	1.1	1.2	1.3	1.4
Hello	Inicia a conexão entre *switch* e controlador.	Simétrica	✓	✓	✓	✓	✓
Error	Indica falha de operação.	Assíncrona	✓	✓	✓	✓	✓
EchoReq	Envia informações sobre latência, largura de banda e se esta ativo.	Simétrica	✓	✓	✓	✓	✓
EchoRes	Resposta ao EchoReq.	Simétrica	✓	✓	✓	✓	✓
Vendor	Utilizado para enviar mensagens proprietárias.	Simétrica	✓				
Experimenter	Mesmo que a Vendor.	Simétrica		✓	✓	✓	✓
FeatureReq	Solicita as capacidades do *switch*.	Crtler⇒*Switch*	✓	✓	✓	✓	✓
FeatureRes	Resposta ao FeatureReq.	Crtler⇒*Switch*	✓	✓	✓	✓	✓
GetConfigReq	Utilizado para questionar e configurar o processamento de pacotes fragmentados.	Crtler⇒*Switch*	✓	✓	✓	✓	✓
GetConfigRes	Resposta ao GetConfigReq.	Crtler⇒*Switch*	✓	✓	✓	✓	✓
SetConfig	Altera o tratamento dos pacotes fragmentados no *switch*.	Crtler⇒*Switch*	✓	✓	✓	✓	✓
PacketIn	Utilizado para enviar pacotes capturados do *switch* para o controlador.	Assíncrona	✓	✓	✓	✓	✓
FlowRemoved	Enviado quando um fluxo é removido no *switch*.	Assíncrona	✓	✓	✓	✓	✓
PortStatus	Envia o status da porta indicada do *switch*.	Assíncrona	✓	✓	✓	✓	✓
PacketOut	Permite injetar pacotes no plano de dados de um *switch*.		✓	✓	✓	✓	✓
FlowMod	Permite que o controlador modifique o estado do *switch*.	Crtler⇒*Switch*	✓	✓	✓	✓	✓
PortMod	Modifica o estado de uma dada porta do *switch*.	Crtler⇒*Switch*	✓	✓	✓	✓	✓
GroupMod	Modifica tabelas de grupos dos *switches*.	Crtler⇒*Switch*		✓	✓	✓	✓
StatsReq	Solicita informações a respeito de um dado fluxo.	Crtler⇒*Switch*	✓	✓	✓		
StatsRes	Resposta ao StatsReq.	Crtler⇒*Switch*	✓	✓	✓		
TableMod	Determinar o destino de pacotes que não possuem entradas em tabelas de fluxos.	Crtler⇒*Switch*		✓	✓	✓	✓
MultipartReq	Mesmo que StatsReq.	Crtler⇒*Switch*				✓	✓
MultipartRes	Resposta ao MultipartReq.	Crtler⇒*Switch*				✓	✓
QueueGetConfigReq	Solicita informações a respeito de filas.	Crtler⇒*Switch*		✓	✓	✓	
QueueGetConfigRes	Resposta ao QueueGetConfigReq.	Crtler⇒*Switch*		✓	✓	✓	
BarrierReq	Utilizado como ponto de sincronização. Assegura que todas as mensagens anteriores tenham sido concluídas.	Crtler⇒*Switch*	✓	✓	✓	✓	✓
BarrierRes	Resposta ao BarrierReq.	Crtler⇒*Switch*	✓	✓	✓	✓	✓
RoleReq	Alterar o papel (mestre/escravo) do controlador frente a vários *switches*.	Crtler⇒*Switch*			✓	✓	✓
RoleRes	Resposta ao RoleRes.	Crtler⇒*Switch*			✓	✓	✓
GetAsysncReq	Solicita quais mensagens assíncronas o *switch* irá enviar.	Crtler⇒*Switch*				✓	✓
GetAsysncRes	Resposta ao GetAsysncReq.	Crtler⇒*Switch*				✓	✓
SetAsysc	Define quais tipos de mensagens podem ser enviadas.	Crtler⇒*Switch*				✓	✓
MeterMod	Modificar a tabela Meter.	Crtler⇒*Switch*				✓	✓
TableStatus	Informa sobre mudanças de estados nas tabelas.	Assíncrona					✓
RequestForward	Informar controladores, quando um dado controlador altera com sucesso grupos e *meters* no *switch*.	Assíncrono					✓
BundleControl	Permite criar, destruir ou efetivar(*commit*) *bundles*.	Crtler⇒*Switch*					✓
BundleAddMessage	Permite adicionar pedidos em um *bundle*.	Crtler⇒*Switch*					✓

Observação: Crtler⇒*Switch* é uma abreviação de controlador-para-*switch*.

Após o estabelecimento do canal de comunicação *switch* e controlador trocam mensagens Hello, que informam sobre a maior versão do protocolo OpenFlow disponível no dispositivo. Enquanto controlador e *switch* estiverem com um canal de comunicação estabelecido esses podem enviar mensagens EchoReq/EchoRes para verificar a largura de banda e latência do canal de comunicação, bem como, identificar se os elementos da rede estão ativos ou não. É bem provável que em uma rede OpenFlow, o *switch* irá enviar para o controlador muitas mensagens Packet-in, pois essa mensagem representa um pacote que foi processado pelas tabelas de fluxo do *switch* e nenhuma entrada foi encontrada para ele, assim o *switch* questiona o controlador sobre como encaminhar esse pacote.

Uma mensagem enviada pelo controlador extremamente importante é a de Packet-out que é uma das respostas à mensagem Packet-in, ou seja, especifica por qual porta do *switch* um dado pacote deve ser encaminhado. Mensagens Packet-out podem ser compostas por um lista de ações ou de nenhuma ação que nesse caso indica que o pacote deve ser bloqueado/descartado. Outras mensagens muito usadas e importantes enviadas pelo controlador são as de mudança de estados (FlowMod, GroupMod, PortMod, MeterMod e TableMod) usadas para adicionar, modificar e alterar entradas nas tabelas, bem como configurar portas do *switch*. Por exemplo, é possível após receber uma mensagem Packet-in usar a programação do controlador para enviar um comando para instalar/adicionar um fluxo específico nas tabelas de fluxo do *switch*, para que esse pacote e seus subsequentes passem por esse novo fluxo e não precisem de mensagens Packet-in e Packet-out para cada pacote que compõem o fluxo. O controlador também pode enviar mensagens PortStatus, StatsReq/StatsRes, MultipartReq/MultipartRes, QueueGetConfigReq/QueueGetConfigRes solicitando estatísticas a respeito do uso de recursos da rede ou ainda FeatureReq/FeatureRes para solicitar características/propriedades dos *switches*. O controlador pode configurar alguns parâmetros do *switch* com as mensagens SetConfig/GetConfigReq/GetConfigRes. Também pode-se fazer uso de mensagens mais elaboradas como as de BarrierReq/BarrierRes que criam um mecanismo de sincronização, já que permite informar o controlador se determinadas ações interdependentes foram executadas no *switch*. Um exemplo seria o envio de duas mensagens, sendo uma para adicionar um grupo x e outra que adiciona um fluxo que utiliza as ações do grupo x. Nesse caso é obrigatório que o *switch* primeiro processe a mensagem que adiciona o grupo x para só depois processar a mensagem que cria o fluxo que usa o grupo x. Caso haja algum problemas na rede OpenFlow, mensagens de erro (Error) podem ser enviadas tanto pelo *switch* quanto pelo controlador.

O *switch* ainda pode enviar mensagens para o controlador informando a respeito da remoção de fluxos ou alteração do status de portas do *switch*. A mensagem FlowRemoved é enviada caso uma entrada da tabela de fluxos seja removida. Isso pode ocorrer porque o tempo dessa entrada expirou ou devido a um pedido de remoção partindo do controlador a partir de uma mensagem FlowMod. O *switch* também envia informações para o controlador caso haja alguma alteração nos status das portas do *switch*. Isso pode ocorrer, por exemplo, caso o cabo de rede seja desconectado da porta do *switch*.

Desde a versão 1.2 do OpenFlow é possível utilizar múltiplos controladores para programar/gerenciar os *switches*. Em um cenário de rede com vários controladores é empregado o esquema de mestre e escravos, dessa forma as mensagens RoleReq/RoleRes são utilizadas para informar qual é o papel (mestre/escravo) de um dado controlador. A partir da versão 1.3 pode-se adicionar filtros para as mensagens assíncronas ou fazer solicitações de informações a respeito de quais mensagens assíncronas os *switches* podem enviar através das mensagens SetAsysc/GetAsysncReq/GetAsysncRes.

O protocolo OpenFlow utiliza um canal de comunicação confiável para a entrega de pacotes, mas não assegura a ordem de processamento dos pacotes e não garante a confirmação automática (*acknowledgement*) se uma mensagem foi processada ou não. A princípio os *switches* devem processar todos os pacotes recebidos e possivelmente enviar uma resposta. Caso ocorra algum problema com o processamento, uma mensagem de erro deve ser emitida.

2.3.1.3 Controlador

O controlador OpenFlow é uma das peças fundamentais das RDS, pois ele representa o plano de controle desacoplado do plano de dados, ou seja, o *switch* agora é responsável somente pelo trabalho de encaminhamento de pacotes e o controlador fica com a tarefa de programar/determinar como o encaminhamento de pacotes será realizado na rede.

Conforme McKeown *et al.* (2008) o controlador OpenFlow pode realizar operações básicas como adicionar e remover entradas nas tabelas de fluxos do *switch* de forma estática, tal como ocorria em um *switch* tradicional, ou executar tarefas mais complexas como permitir que os fluxos de rede sejam criados e características dos *switches* sejam configuradas dinamicamente para atender as necessidades dos usuários/administradores da rede. Em Lara *et al.* (2013) e Nunes *et al.* (2014) o

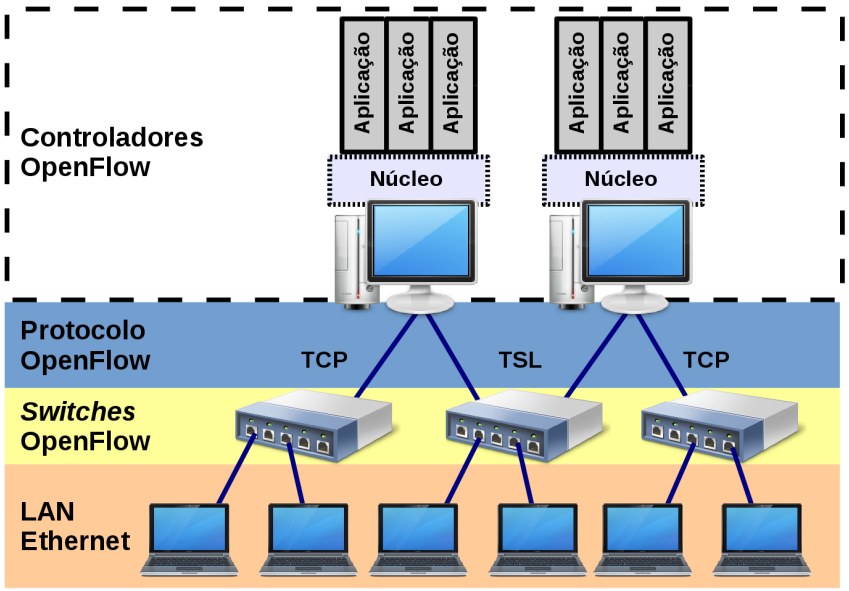


Figura 2.12: *Controlador OpenFlow (Flowgrammable, 2014; Nunes et al., 2014)*

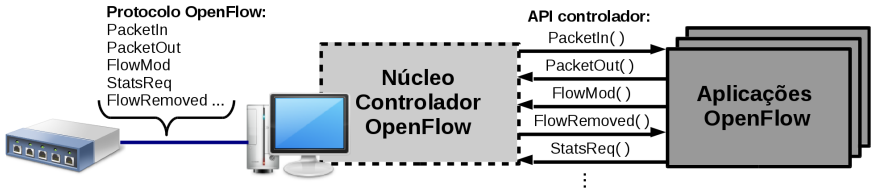


Figura 2.13: *Fluxo de mensagens entre switch, controlador e aplicações (Flowgrammable, 2014)*

controlador OpenFlow é ilustrado como uma interface de programação que pode ser utilizada para gerenciar e implementar novas funcionalidades na rede. Os autores comparam o controlador OpenFlow a um sistema operacional de rede (Tanenbaum, 2007), ou seja, esse oferece uma interface de comunicação/gerenciamento dos recursos da rede o que permite construir mais facilmente aplicações que fazem uso dos recursos disponíveis da rede.

O controlador pode instalar fluxos nos *switches* de forma reativa ou pró-ativa. Quando o controlador trabalha de forma reativa significa que os comandos serão enviados conforme os eventos forem ocorrendo na rede, já no modo proativo o controlador antecipa os eventos e por exemplo, instala os fluxos antes que sejam requeridos pelo *switch* (ONF, 2013). Uma rede OpenFlow, também pode ter apenas um único controlador o que representa um esquema centralizado para o plano de controle. Outras possibilidades é ter um controlador por *switch* em um esquema descentralizado, ou ter várias configurações de *switch* e controlador, tal como, dois controladores para um grupo de *switches*.

A Figura 2.12 apresenta a arquitetura genérica do controlador OpenFlow, bem como sua interação com os demais elementos da rede. O núcleo é a principal parte do controlador e pode ser visto como um sistema operacional de rede, pois tem por função identificar e gerenciar eventos que estejam ocorrendo na rede e mapear as mensagens do protocolo OpenFlow em funções ou objetos que ficam disponíveis para a programação da aplicação, tal como ilustrado na Figura 2.13. As aplicações OpenFlow são justamente uma das grandes inovações das RDS e um controlador pode executar várias aplicações, sendo que cada aplicação pode exercer uma tarefa distinta na rede, como por exemplo: realizar o serviço de comutação tradicional; encaminhamento de pacotes inter-VLANs; tratar de problemas relacionados a QoS; segurança; e assim por diante.

Atualmente existem vários arcabouços de controlador OpenFlow. A tabela 2.4 apresenta alguns desses controladores. Basicamente os arcabouços permitem controlar os recursos dos *switches* Open-

Flow como: portas, tabelas, fluxos, etc, através de mensagens OpenFlow, tal como já foi detalhado nas Seções 2.3.1.1 e 2.3.1.2. Há mais arcabouços do que os apresentados na Tabela 2.4. Na maioria dos casos eles se diferenciam pela linguagem de programação e pela especialidade em algum determinado assunto (amigabilidade, roteamento, QoS, etc). Muitos dos arcabouços de controladores são de código aberto (*open source*). Dentre os controladores destacam-se o NOX (NOX, 2014), que foi o primeiro controlador OpenFlow implementado. Dele surgiu o POX (POX, 2014) que é um controlador muito utilizado em pesquisas acadêmicas devido a sua programação em Python, que permite criar rapidamente protótipos que representam novas formas de controlar a rede.

Tabela 2.4: *Arcabouços OpenFlow*

Controlador	Linguagem	Desenvolvedor	Descrição
NOX	Python/C++	Nicira	Foi o primeiro controlador OpenFlow desenvolvido. Foi doado pela Nicira para a comunidade acadêmica (NOX, 2014).
POX	Python	Nicira	Derivado do NOX, possui plataforma simples de programar e por isso é muito utilizado para o desenvolvimento de protótipos. Muito utilizado em pesquisas e para fins educacionais (POX, 2014).
Ryu	Python	NTT, OSRG group	Ryu é um arcabouço de RDS que suporta vários protocolos tal como o OpenFlow (Ryu, 2014).
Flowvisor	C	Standord/Nicira	É um tipo especial de controlador que atua como *proxy* entre os *switches* e múltiplos controladores OpenFlow com o objetivo de dividir a rede e delegar a cada controlador parte da rede que foi dividida (Flowvisor, 2014).
SNAC*	C++	Nicira	Baseado no NOX-0.4, utiliza interface Web amigável para o gerenciamento e configuração de políticas da rede (SNAC, 2014).
RouteFlow	C++	CPQD	Controlador que integra soluções legadas de roteamento IP com OpenFlow (RouteFlow, 2014).
Maestro	Java	Rice University	Sistema Operacional de rede que fornece interface para modular a implementação de aplicações que controlem os recursos da rede (Maestro, 2014).
Beacon	Java	Stanford	O Beacon é considerado um controlador rápido (Erickson, 2013), multiplataforma que suporta operações baseadas em eventos e *threaded* (Erickson, 2014).
Floodlight	Java	BigSwitch	Baseado no Beacon e suporta uma grande gama de *switches* virtuais e reais (Floodlight, 2014).

* - O SNAC possui licença proprietária.

Outro arcabouço que merece destaque é o Beacon (Erickson, 2014) que foi desenvolvido na Universidade de Stanford - uma das precursoras do OpenFlow. O Beacon também é a base de outros controladores, como por exemplo o Floodlight. No trabalho (Erickson, 2013) são apresentados bons resultados de desempenho do Beacon em relação a outros controladores, principalmente quando o hardware do controlador dá suporte a várias linhas de execução (*threads*), pois o Beacon é totalmente fundamentado em processamento paralelo através de múltiplas linhas de execução (*multithread*), assim tanto o núcleo (*core*) da arquitetura do Beacon quanto as aplicações são executadas em múltiplas linhas de execução, o que torna possível tratar simultaneamente várias mensagens de *switches*, bem como executar paralelamente as aplicações OpenFlow. Outra característica marcante do Beacon é implementação modular que dá a possibilidade de ligar, desligar ou reiniciar aplicações OpenFlow sem a necessidade de interromper os serviços básicos do controlador, essa característica torna o Beacon muito mais parecido com um sistema operacional de rede, já que normalmente sistemas operacionais multitarefa apresentam tal comportamento (Tanenbaum, 2007).

A Figura 2.14 ilustra a arquitetura do controlador Beacon. É possível observar que toda arquitetura é fundamentada em um núcleo que é responsável pelas tarefas básicas do controlador e as aplicações são executadas sobre esse núcleo. No Beacon existem algumas aplicações que fazem parte do arcabouço, essas são:

- *Learning Switch*: aplicação que implementa o esquema de comutação de pacotes tradicional em *switches* Ethernet;
- *Device Manager*: responsável por manter informações a respeito de dispositivos conectados a rede. Tais informações são úteis ao gerenciamento da rede. São exemplos dessas informações:

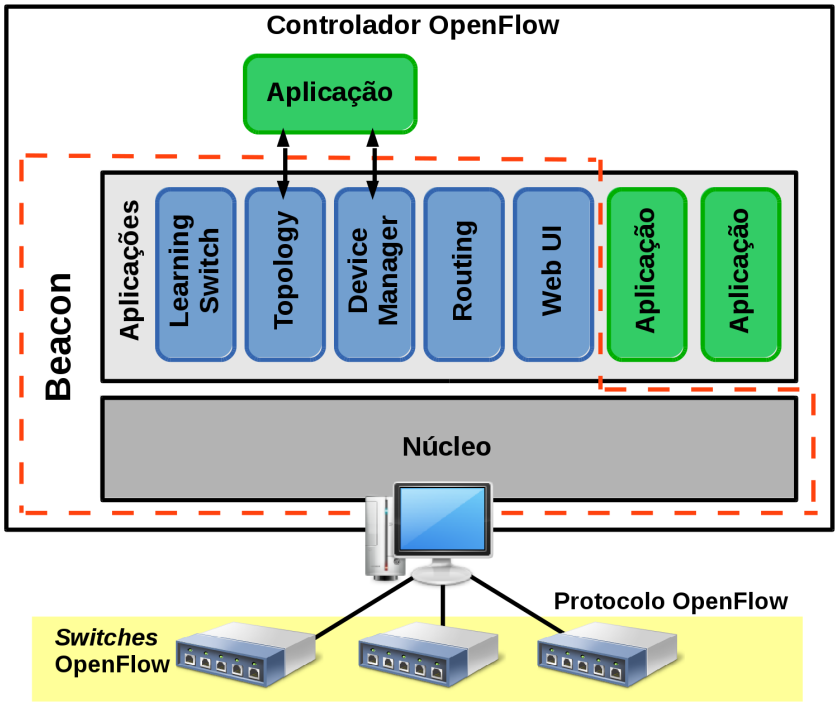


Figura 2.14: *Arquitetura do controlador OpenFlow Beacon (Erickson, 2013)*

endereços do dispositivo, última data de acesso, portas utilizadas, etc;

- *Topology*: utilizado para descobrir enlaces (*links*) de ligação entre *switches* e armazenar informações sobre essas ligações (portas). Essa aplicação pode ser útil caso seja necessário restabelecer ligações em caso de falhas ou para evitar laços (*loops*) entre os *switches*;
- *Routing*: usado para calcular o caminho mais curto entre dispositivos da rede. Para realizar esse cálculo, essa aplicação, faz uso das aplicações Device Manager e Topology;
- *Web* UI: provê interface Web que permite, por exemplo, visualizar dispositivos e eventos da rede.

Além das aplicações que estão disponíveis junto com o arcabouço também é possível implementar outras aplicações usando funções básicas do núcleo, bem como desenvolver aplicações utilizando funções disponíveis a partir de outras aplicações já existentes.

Um exemplo de codificação utilizando o arcabouço Beacon é ilustrado na Figura 2.15, no código em questão o controlador faz com que o encaminhamento de pacotes no *switch* seja feito tal como faria um *hub*, ou seja, qualquer pacote enviado para o controlador é difundido para todas as portas do *switch*, o que também representa uma inundação (*flood*) na comutação utilizada em *switches* tradicionais. O método `forwardAsHub()` tem como parâmetro objetos que representam o *switch* (`sw`) e o pacote que foi enviado através de uma mensagem OpenFlow PacketIn (`pi`) e será analisado pelo controlador. A resposta do controlador ao *switch* se dará via mensagem OpenFlow PacketOut, por isso na linha 2 é criado o objeto `po`. Na linha 3 é instanciado o objeto `action` do tipo `OFAction` que representa a ação que será aplicada ao pacote, nesse caso a ação é enviar o pacote para todas portas do *switch*, exceto a porta que o pacote adentrou no *switch*, isso é feito utilizando o comando `OFPort.OFPP_FLOOD.getValue()` e tal ação é relacionada à mensagem OpenFlow de PacketOut na linha 4. Nas linhas 5 e 6 utilizando as informações contidas em `pi`, o objeto `po` ainda tem configurado qual é a porta de entrada e principalmente a informação de onde está armazenado o pacote na memória (*buffer*) do *switch* para que esse possa ser devidamente encaminhado - retirado da memória do *switch* e enviado à(s) porta(s) de destino. Contudo, caso o pacote não tenha sido

Figura 2.15: *Exemplo de método que implementa um hub utilizando o arcabouço Beacon (Erickson, 2014)*

```
1 public void forwardAsHub(IOFSwitch sw, OFPacketIn pi) throws IOException {
2   OFPacketOut po = new OFPacketOut();
3   OFAction action = new OFActionOutput(OFPort.OFPP_FLOOD.getValue());
4   po.setActions(Collections.singletonList(action));
5   po.setInPort(pi.getInPort());
6   po.setBufferId(pi.getBufferId());
7   if (pi.getBufferId() == OFPacketOut.BUFFER_ID_NONE) {
8         po.setPacketData(pi.getPacketData());
9   }
10  sw.getOutputStream().write(po);
11 }
```

armazenado na memória do *switch*, esse deve ser enviado diretamente pelo controlador dentro da mensagem PacketOut, isso é feito nas linhas 7-9. Por fim, a mensagem PacketOut é enviada para o *switch* através do comando da linha 10. Esse exemplo demostra como se dá o tratamento dos pacotes de rede dentro de um controlador OpenFlow e como é possível programar o plano de dados através das interfaces fornecidas pelos arcabouços OpenFlow.

Capítulo 3

Trabalhos relacionados

Neste capítulo são apresentados os trabalhos relacionados com a área de cibersegurança e que também utilizam técnicas de CA.

Desde a idealização da CA no ano de 2001 (IBM, 2003; Kephart e Chess, 2003), diversos trabalhos vem empregando técnicas autonômicas para melhorar o nível de segurança de ambientes computacionais. A seguir são apresentados trabalhos que utilizam CA para fornecer segurança para as mais diversas finalidades dentro do âmbito computacional.

Um dos primeiros trabalhos que relacionam CA e segurança foi apresentado por Chess *et al.* (2003). Os autores afirmam que a segurança é parte vital para soluções autonômicas, pois sistemas autonômicos podem ser compostos por vários elementos interconectados através de diferentes domínios administrativos, o que acrescenta novos desafios à segurança. Por exemplo, um elemento pode se auto-configurar inapropriadamente e de forma insegura para conseguir estabelecer comunicação com outros elementos autonômicos. Além de abordar novos problemas de segurança que os sistemas autonômicos podem apresentar, Chess *et al.* (2003) também afirmam que o uso de CA pode ajudar a proteger ambientes informatizados, o que por exemplo, é relatado em outro trabalho (Chess , 2005), que discute abordagens de CA no combate a vírus. Segundo os autores, um dos principais motivos para usar CA aplicado a segurança é que os ambientes computacionais, bem como os ataques à segurança, estão se tornando cada vez mais complexos e elaborados, o que dificulta muito as soluções baseadas em reações humanas. Dessa forma, o uso de CA é uma escolha apropriada, pois dispensa ou minimiza a intervenção humana no combate a problemas de segurança. (Chess *et al.*, 2003) apresentam vários problemas e soluções de segurança que utilizam técnicas autonômicas, por isso esse trabalho é a base de outros trabalhos que serão apresentados a seguir.

Os trabalhos (Srivastava e Sahu, 2004), (Zhou e Foley, 2004), (Kayem *et al.*, 2008), (Abie, 2009) abordam o uso de CA para auto-proteger informações armazenadas ou em trânsito e auto-gerenciar configurações de protocolos relacionados com a segurança do sistema. De uma forma geral esses trabalhos mensuram o nível de segurança através da observação do ambiente e respondem às mudanças ajustando parâmetros internos, como esquemas de criptografia, configurações de protocolos de segurança, mecanismos de autenticação e QoS.

Também existem trabalhos que utilizam CA para fornecer segurança no controle de acesso a recursos do sistema. Nesse contexto, Aljnidi e Leneutre (2007) introduzem um arcabouço para fornecer segurança para redes autonômicas móveis utilizando conceitos de comunicação confiável, autenticação baseado em categorização e relação de capacidades entre os *hosts* da rede. Nos trabalhos (He *et al.*, 2010a) e (He *et al.*, 2010b) os autores apresentam a arquitetura VSK (*Virtual Security Kernel*) que implementa estruturas de autorização para aplicações em nível de sistema operacional. Já Bailey (2012) propõe o arcabouço SAAF (*Self-Adaptive Authorization Framework*) para autorização auto-adaptável em ambientes federalizados, esse arcabouço é capaz de monitorar o uso de recursos e controlar acessos utilizando políticas de autorização, direitos de acessos, sessões, dentre outros.

Os autores Ananthanarayanan *et al.* (2005) e Yau *et al.* (2006) abordam alguns conceitos de engenharia de software para respectivamente tratar: de conflitos de contratos entre elementos/sis-

temas autonômicos que precisam de interação com outros sistemas; e do uso de um modelo de alto nível que abstrai o desenvolvimento de softwares complexos separando requisitos de segurança de detalhes técnicos envolvidos na implementação.

Em (Alomari e Menasce, 2012) e (Alomari e Menascé, 2013) os autores utilizam CA para fornecer segurança para sistemas de gerenciamento de banco de dados, sendo um dos objetivos equalizar segurança e QoS.

Wang e Ma (2007) propõem um modelo de auto-monitoramento e auto-proteção para tratar do problema de *spam* em motores de busca. O auto-monitoramento se dá pela observação de resultados de busca que podem ser *spam*, então um sistema de auto-proteção é ativado para confirmar se os resultados suspeitos são ou não frutos de *spam*.

Maliki e Seigneur (2010) apresentam um método autonômico para equalizar o consumo de energia de dispositivos móveis levando em consideração o nível de segurança exigido por cada dispositivo em diferentes contextos de usuários e aplicações. Os autores alegam que ao se considerar contextos de segurança em dispositivos móveis é preciso levar em consideração a energia utilizada, pois quanto mais segurança é aplicada, mais energia é gasta. Desta forma é possível desligar módulos de segurança caso o dispositivo não esteja sendo atacado e assim conservar bateria.

Somayaji *et al.* (2008) apresentam sugestões e críticas em uma análise do uso de sistemas bioinspirados aplicados a segurança, o que inclui CA. O trabalho afirma que para se utilizar conceitos da biologia é necessário identificar propriedades de sistemas vivos que podem ser replicadas em sistemas digitais, entender como os sistemas vivos mantém essas propriedades e traduzir a biologia para mecanismos computacionais. Outro ponto questionado é que existem propriedades da segurança da informação que não são contempladas pela biologia, tal como a confidencialidade dos dados já que nenhum organismo vivo tenta esconder informações. O objetivo de organismos vivos é maximizar a sua disponibilidade, ou seja, é manter-se vivo. Ainda na biologia, os atacantes são formas de vida simples sem muita inteligência, já na computação os atacantes são muitas vezes pessoas com grande capacidade intelectual, o que pode dificultar muito a detecção e remediação do ataque.

Os trabalhos apresentados anteriormente dão um panorama do uso de CA para manter a segurança com os mais diversos propósitos. A seguir serão apresentados trabalhos que estão mais relacionados com a proposta do presente trabalho, que é empregar CA para manter a segurança de redes de computadores locais. Para um melhor entendimento os trabalhos foram separados por subáreas, sendo essas: políticas de segurança, detecção de anomalias; negação de serviço; e sistemas de detecção de intrusão.

3.1 Políticas de segurança

Wan e Alagar (2006) apresentam uma arquitetura autonômica que permite alterar a política de segurança levando em consideração o contexto do usuário, tal como localização do usuário, horário de acesso ao sistema, dentre outras características.

Shen *et al.* (2009) apresentam brevemente o arcabouço ISDS (*Intelligent Security Defensive Software*), que pode adaptar-se ao ambiente mudando as políticas de segurança e cooperando com outros softwares de segurança como *firewall* e anti-vírus para criar uma barreira de defesa contra ataques. O ISDS utiliza um modelo baseado em monitoramento/análise/planejamento/execução e possui uma base de conhecimento que mantém informações que são utilizadas em tomadas de decisões. Nesse arcabouço a base de conhecimento deve ser alimentada pelas informações do sistema e pelo administrador.

Barrère *et al.* (2011) afirmam que as redes autonômicas podem gerar novas vulnerabilidades e desta forma propõem um modelo para traduzir descrições de vulnerabilidades de elementos autonômicos em regras para políticas de segurança. Tais regras são interpretadas por um sistema autonômico que deve detectar vulnerabilidades e executar operações para sanar o problema. A descrição das vulnerabilidades em redes autonômicas se dá através da extensão da linguagem OVAL (*Open Vulnerability Assessment Language*). O repositório OVAL fornece uma variedade de descrições de vulnerabilidades, que podem servir como fonte de informações para a detecção de problemas

de segurança. Em outro trabalho (Barrère *et al.*, 2012), os autores dão continuidade à pesquisa, apresentando a linguagem DXCCDF (*Distributed eXtensible Configuration Checklist Description Format*), que tem por objetivo descrever configurações de vulnerabilidades em ambientes distribuídos e assim permitir que medidas sejam tomadas conforme as vulnerabilidades sejam detectadas. Porém o sistema de Barrère *et al.* (2011) e Barrère *et al.* (2012) só emite alertas e não reage frente às ameaças. Em (Barrère *et al.*, 2014), os autores apresentam alguns desafios a serem superados no que se refere a área de CA relacionada com cibersegurança.

Sibai e Menasce (2011) afirmam que as ameaças à segurança mais devastadoras são as internas e não as externas. Assim, propõem um arcabouço chamado AVPS (*Autonomic Violation Prevention System*), que tem por objetivo a auto-proteção contra violações de regras da política de segurança da rede. O AVPS utiliza um NIDS com políticas personalizadas para proteger sistemas de abusos de usuários com privilégios ou de configurações mal elaboradas por parte dos administradores. Essa proteção é dada em forma de políticas que são descritas como regras implementadas através do Snort e aplicadas em todo o sistema pelo AVPS. As regras basicamente informam qual endereço de IP pode acessar qual serviço ou se um acesso a rede pode ou não utilizar comandos como: *ls*, *pwd*, *rm *.log*, *vi*. Na abordagem proposta pelos autores, todas as regras que vem por padrão no Snort são desabilitadas/removidas e substituídas por regras que representam a política de segurança da rede. A criação dessas regras é tarefa do administrador da rede. Portanto, com essa abordagem o Snort não detecta ataques externos. As principais características do AVPS são: 1) Detectar violações na política e não intrusão; 2) Escalabilidade, pois a política deve afetar muitos *hosts*; 3) Fácil gerenciamento, pois uma política maior é aplicada em muitos *hosts*; 4) Fácil manter, pois uma regra criada no AVPS é aplicada em vários *hosts*; 5) Força a separação de deveres, pois o administrador do sistema tem controle apenas sobre o seu sistema e o administrador da rede tem controle sobre toda a rede. Em outro trabalho (Sibai e Menasce, 2012), os autores expandem o AVPS fazendo com que regras de alto nível sejam criadas de forma dinâmica. Tais regras disparam ações e eventos em baixo nível visando bloquear ameaças internas ao sistema. O objetivo é minimizar a tarefa de criação das regras do administrador. Experimentos apresentados em Sibai e Menasce (2012) mostram que é possível gerar regras automáticas através de pacotes de rede com um baixo número de falsos positivos.

Wailly *et al.* (2012) propõem uma arquitetura chamada VESPA (*Virtual Environments Self-Protecting Architecture*) que provê auto-proteção em ambientes de nuvens (*cloud computing*). Segundo os autores, ambientes de nuvem são compostos por vários níveis de software, que são independentes e possuem problemas de segurança distintos. Para fornecer auto-proteção são aplicadas políticas, pois essas normalmente atentem bem requisitos de segurança em ambientes heterogêneos e multicamadas, tal como um ambiente de nuvem. A arquitetura proposta é baseada em camadas. Na camada 1 estão os recursos a serem monitorados e protegidos. Na camada 2 existe um plano de segurança para elementos de detecção e reação a ameaças. A camada 3 tem por função abstrair detalhes, tal como heterogeneidade e fazer intermédios entre as camadas. A camada 4 é responsável pela orquestração e toma decisões em mais alto nível, já que possui a visão geral da nuvem. A auto-proteção é dada através de um conjunto de ciclos autonômicos operando sobre componentes que compõem o ambiente de nuvem. Assim, a VESPA regula a proteção de recursos através da coordenação autonômica que monitora diferentes camadas de infraestrutura.

3.2 Detecção de anomalias

Marnerides *et al.* (2008) propõem uma arquitetura baseada em medidas resilientes para prover auto-proteção e auto-otimização em redes de computadores. A arquitetura utiliza informações de um módulo de monitoramento que coleta pacotes de redes e envia para um motor de detecção. Os pacotes geram fluxos de redes que são agrupados e classificados como normal ou anormal através de supervisão por um classificador *Naive Bayes*. Caso sejam identificados fluxos anormais, um motor de remediação deve decidir como combater ameaças, por exemplo, podem ser aplicadas técnicas de balanceamento de carga. A arquitetura também aprende sobre os padrões de pacotes que chegam na

rede, para isso uma unidade notificadora atualiza informações no motor de remediação caso sejam diagnosticados ataques.

Varas e Hirsch (2009) apresentam o D-CAF (*Distributed Context-Aware Firewall*) que é uma abordagem para proteger serviços bem conhecidos/comuns de um dado ambiente de rede. O objetivo não é identificar ataques, mas só mitigar exageros de serviços ou usuários utilizando a rede, e quando isso ocorrer manter os serviços básicos sempre funcionando. Para conhecer os serviços em uma rede o D-CAF pontua tais serviços baseando-se em características como o número de acessos a páginas Web, se a página Web requer ou não autenticação, dentre outras. Essas observações são enviadas para o núcleo do D-CAF, que irá cruzá-las com informações de roteamento e outras estatísticas da rede para evidenciar anomalias. No caso de sobrecarga na rede o D-CAF utiliza um *firewall* para bloquear serviços com baixa pontuação.

Zhang e Shen (2009) apresentam um arcabouço estatístico que permite caracterizar e analisar detectores de anomalia observando propriedades operacionais. O arcabouço é utilizado para o desenvolvimento de um *middleware* adaptativo chamado M-AID (*Middleware for Anomaly-based Intrusion Detection*), que é empregado em *hosts* e permite ajustes automáticos através de algoritmos de reforço de aprendizado. O M-AID utiliza o conceito de multiagentes e integra vários detectores de anomalia usando o processo de decisão de Markov. Desta forma, o M-AID deve se auto-adaptar a diversos ambientes operacionais. O *middleware* proposto fornece ainda uma interface que permite ao administrador observar o contexto de segurança do sistema.

Guan e Fu (2010) apresentam o Auto-AID (*Autonomic Anomaly IDentification*) que tem por objetivo identificar anomalias em sistemas computacionais de grande porte. Conforme os autores, ambientes computacionais compostos por inúmeros *hosts*, tal como uma grade computacional, possuem enormes montantes de dados a serem processados durante o processo de identificação de problemas. Para amenizar tal problema, Guan e Fu (2010) empregam mineração de dados. No Auto-AID os dados relacionados à saúde do sistema são coletados e enviados para análise, que inclui: transformação de dados, extração de características, agrupamento e identificação de anomalias. Técnicas de redução de dimensionalidade são empregadas para reduzir a quantidade de dados a serem processados em busca de anomalias. Também a distância Euclidiana é utilizada para verificar dissimilaridade, ou seja, se o comportamento de um *host* é significativamente diferente dos demais, o que seria considerado anormal. Quando eventos suspeitos são identificados os mesmos são reportados para os administradores, para o tratamento apropriado.

Ippoliti e Zhou (2012) propõem uma arquitetura para detecção de anomalias em redes autonômicas. A arquitetura é composta por um controlador que deve fornecer auto-otimização e auto-ajustes utilizando para isso parâmetros de desempenho e prioridades que são a princípio passados pelo administrador da rede e posteriormente mantidos autonomicamente pelo controlador através do emprego de técnicas de reforço de aprendizado e redes neurais. Na arquitetura proposta o tráfego de rede passa por um pré-processador de tráfego que gera estatísticas que são usadas para atualizar os parâmetros do sistema para manter, por exemplo, a performance do sistema, bem como são enviadas para um módulo de detecção de anomalia. Caso sejam identificados problemas na rede, são empregadas medidas para tratar o problema, tais medidas podem ser desde a aplicação de prioridades chegando até ao bloqueio do fluxo de rede.

3.3 Negação de serviço

Loukas e Oke (2007) detectam ataques DoS utilizando um sistema inspirado na biologia que combina decisões *Bayesianas* e redes neurais aleatórias. São mensuradas inúmeras variáveis estatísticas e instantâneas que descrevem o tráfego de rede e as saídas são fundidas usando-se redes neurais para decidir se um fluxo de rede é normal ou representa um ataque.

Berral *et al.* (2008) apresentam uma arquitetura distribuída e adaptativa para detecção antecipada e contenção de ataques de inundação e uso abusivo de redes. Nessa arquitetura um dado *host* identifica e reage a ataques de DoS através da análise do seu próprio fluxo de rede, bem como do fluxo de rede dos *hosts* vizinhos, o que deve fornecer uma visão global do comportamento da rede.

Assim, um subconjunto de *hosts* da rede são equipados com detectores e classificadores que analisam a quantidade de pacotes vindos de um dado *host* para distinguir se a rede está sofrendo ataques de DoS ou não. O método *Naive Bayes* é empregado para integrar informações globais e locais dos *hosts* e o algoritmo CUSUM (*Cumulative Sum*) é utilizado para identificar se o tráfego de rede apresenta alterações significativas ou não. Caso um ataque seja identificado, são emitidos alertas pelo *host* sob ataque e pelos vizinhos. Então, os *hosts* próximos da vítima irão parar o ataque, por exemplo, o alerta chega a um roteador que bloqueia a origem do ataque. O ideal é parar o ataque o mais próximo possível da sua origem. Como cada *host* possui características diferentes torna-se um problema configurá-los com precisão. O mecanismo foi testado e produz bons resultados, pois conseguiu identificar com 95% de precisão os ataques e assim impedir ataques nas bordas da rede.

de Azevedo *et al.* (2009) propõem o uso de um modelo autonômico chamado AutoCore, que é constituído por um sistema multiagente, uma interface inteligente chamada CoreEditor e uma ontologia chamada CoreSec. Esse modelo visa apoiar o processo de segurança em ambientes corporativos através de um modelo formal que reflete os aspectos de segurança do ambiente, permitindo comunicação segura entre os vários elementos do sistema autonômico e possibilitando também analisar os diferentes eventos que podem afetar a segurança do sistema. O AutoCore faz uso do CoreSec que é uma base de conhecimento de ontologias sobre segurança da informação. A arquitetura do AutoCore por sua vez é formada por um Agente Gerente que é responsável pela auto-configuração, ou seja, por definir os objetivos e tarefas a serem executadas pelos outros agentes. Ainda existem agentes responsáveis por analisar e recuperar informações sobre tráfego de rede; enviar o status do sistema; encontrar a melhor ação a ser tomada; executar ações para corrigir problemas; mostrar informações e gráficos para que o responsável pela segurança possa acompanhar o estado do ambiente; e alimentar a base de conhecimento usada pelo AutoCore. O sistema foi submetido a testes com ataques de DoS e os autores concluíram que o modelo se saiu bem sob esse tipo de ataque.

Rawat e Saxena (2009) propõem uma abordagem autonômica para detecção de ataques DoS chamada de teoria do perigo (*danger theory*). A abordagem é inspirada no sistema de imunidade humana, mais especificamente na ideia de que quando algo estranho adentra ao organismo e mata prematuramente alguma célula, medidas reativas entram em ação para combater tal organismo. Se um sinal de perigo for identificado o sistema deve investigar os dados, aprender a assinatura da atividade maliciosa e combater o problema. No trabalho proposto por Rawat e Saxena (2009) o sinal de perigo é emitido quando a CPU está sobrecarregada e isso é atribuído a uma grande quantidade de pacotes TCP, com o campo de controle SYN ativo, chegando ao sistema. Nesse cenário computacional conclui-se que o sistema está sofrendo um ataque DoS e um alerta deve ser emitido ao ambiente computacional. Na proposta, o sistema deve suportar/sofrer o primeiro ataque, depois o sistema vai aprender com esse e não sofrerá mais com tal ataque. Assim, a teoria do perigo cria um sistema baseado em reação, caso haja uma reação ruim, o corpo reage contra o elemento estranho, caso contrário não, o que deve gerar poucos falsos positivos.

3.4 Sistemas de detecção de intrusão

Armstrong *et al.* (2003) apresenta um sistema de defesa autonômica que identifica processos maliciosos em um dado *host*. O objetivo do sistema é reagir automaticamente a ataques que estão ocorrendo e evitar falsos positivos. O sistema proposto é composto por um mecanismo estocástico de aprendizado (*feedback*) baseado no processo de decisão de Markov. O controlador obtém entradas de sensores de anomalia disponíveis comercialmente, calcula a probabilidade do sistema estar sob ataque e aplica ações em atuadores para reagir aos ataques. O processo de decisão de Markov, aplicado em um simulador *off-line*, deve garantir que o sistema possua observabilidade, ou seja, tenha capacidade de determinar o estado verdadeiro do sistema e controlabilidade, que é a habilidade do controlador afetar o estado do sistema. Após testes os autores chegaram a conclusão de que é muito difícil relacionar o comportamento do controlador com os ataques.

Qu *et al.* (2004) apresenta um arcabouço de monitoramento e análise para prover auto-proteção contra ataques de redes. Para isso os autores utilizam softwares agentes que monitoram vários

atributos dos elementos da rede para caracterizar seus respectivos estados como normal, incerto ou anormal. Os atributos monitorados podem ser *e-mails*, pedidos ARP, etc. O software agente também possui mecanismos de recuperação para tentar restaurar o estado normal do elemento da rede.

Merideth e Narasimhan (2005) abordam como utilizar CA em aplicações de redes legadas, mais especificamente em NIDS. Os autores citam como problema que muitos programas legados não são feitos para serem autonômicos. Por exemplo, o Snort não é autonômico, mas seria interessante se fosse, pois um NIDS não deve permanecer com uma única configuração estática e sim ter configurações dinâmicas para responder às novas ameaças que surgem diariamente e também cada ambiente computacional requer uma configuração de NIDS distinta e dinâmica, já que o ambiente de rede muda constantemente, assim não existe uma configuração padrão e estática que atenda de forma ótima todos os ambientes. Desta forma Merideth e Narasimhan (2005) propõem uma adaptação no Snort para permitir que o NIDS se auto-configure utilizando para isso atualizações disponíveis na Internet e informações a respeito do ambiente de rede. Para essa tarefa o trabalho utiliza o Flexiphant que é um mecanismo que reúne parâmetros de configuração a respeito da rede para gerar novas regras de configuração no Snort. Para obter informações sobre a rede o Flexiphant utiliza dados que são publicados por *hosts* na rede através de um software agente, faz varredura (*scan*) em *hosts* da rede e armazena todas essas informações em um repositório. De posse das informações dos *hosts* o Flexiphant pode habilitar, desabilitar e mudar prioridades de regras do Snort, o que torna a configuração desse NIDS autonômica.

Shi *et al.* (2005) apresentam uma arquitetura que utiliza sistemas multiprocessados para prover tolerância a falhas e auto-recuperação em caso de problemas de segurança em aplicações. Para isso os núcleos da arquitetura de múltiplos processadores trabalham com dois níveis de privilégios: monitor e protegido. O monitor tem alto nível de privilégio e garante acesso a todos recursos de hardware. O modo protegido possui poucos privilégios e acessa os recursos de hardware de forma controlada e limitada. Os modos monitor e protegido trabalham isolados e podem executar diferentes sistemas operacionais virtualizados. Aplicações comuns são executadas no modo protegido e caso tentem fazer acesso a áreas não permitidas, passam a ser tratadas pelo modo monitor. Quando um problema de segurança é identificado pelo modo monitor esse pode controlar vários aspectos do núcleo protegido, podendo por exemplo: reiniciar/recuperar o núcleo protegido; limpar *pipelines*; etc. Dessa forma, no caso de anomalias é possível voltar o sistema a um ponto (*checkpoint*) em que não existiam problemas. No entanto só é possível recuperar poucos estados e não é possível reverter alterações no sistema de arquivos. O sistema proposto consegue monitorar vários tipos de *rootkits* e estouros de pilhas, mas não consegue resolver ataques de DoS.

Arora *et al.* (2006) apresentam um estudo que usa teoria dos jogos para comparar o desempenho de IDSs que: não utilizam CA; empregam CA parcialmente; e que fazem uso pleno de CA. O objetivo foi analisar economicamente se compensa ou não usar CA como ferramenta de segurança. Arora *et al.* (2006) concluíram que o uso parcial de CA em IDS é recomendável em ambiente onde é custoso fazer a análise manual do ataque e existe pouco tráfego de usuário. Já uma abordagem totalmente autonômica é recomendável em ambientes com grande tráfego de usuário, no qual o comportamento do ambiente muda constantemente, pois diminui o potencial de ataques e reduz o problema de falsos positivos, o que claramente é visto como uma melhora em relação a investigação manual de ataques à segurança.

Dai *et al.* (2006) apresentam um mecanismo autonômico de segurança baseado em neurônios virtuais distribuídos para reconhecimento de padrão. Os neurônios são softwares instalados nos *hosts* e são compostos por três partes: coletor de informação, mecanismo de comunicação com o neurônio vizinho e mecanismo de reconhecimento de padrões. O mecanismo de comunicação trabalha de forma P2P e com estrutura hierárquica. Caso algum fluxo ou mensagem seja identificada como ilegal são utilizados dois passos: 1) a origem da mensagem/fluxo é rastreada e identificada; 2) a mensagem/fluxo pode ser bloqueada antes de alcançar o destino. Durante os testes constatou-se que os neurônios virtuais conseguem detectar ataques de: *eavesdropping*, *replay*, *masquerading*, *spoofing* e DoS.

Chiang *et al.* (2007) acreditam que a adaptabilidade dos sistemas autonômicos pode influenciar negativamente na segurança das redes, assim os autores tentam achar através de um sistema bio-inspirado um ponto de equilíbrio entre adaptabilidade e segurança. Para atender a esse objetivo Chiang *et al.* (2007) propõem o sistema VAS (*Vulnerability Awareness System*) e o O:MIB (*Object-oriented Management Information Base*), nesse contexto o VAS deve encontrar o limiar mínimo entre eficiência e segurança utilizando para isso os dados da O:MIB. A avaliação de vulnerabilidades proposta, dá apenas indícios de ataques e não certezas. Durante os testes foram analisadas as mudanças nas informações do O:MIB, as condições de tráfego de rede e distúrbios/erros ocorridos.

Al-Nashif *et al.* (2008) apresentam o sistema ML-IDS (*Multi-Level Intrusion Detection System*), que é um IDS multi-nível que utiliza CA. O ML-IDS detecta e reage a ataques de redes analisando estados dos protocolos de rede, inspeção de cabeçalhos e carga útil dos pacotes. Análise de comportamento é baseada em regras (*rule-based*) que são geradas usando aprendizagem supervisionada para detectar ataques. Os módulos de análise enviam informações para um módulo de fusão, que tem por objetivo aumentar a taxa de detecção, combinando as análises e reduzindo falsos positivos. Depois, o módulo de fusão envia as informações para dois motores: risco e impacto, que devem respectivamente determinar como o sistema será afetado pelo ataque e como uma dada contramedida irá afetar o sistema. Depois de decidir o que fazer o motor de fusão envia as ações a serem tomadas para o motor de ação, que previne ou reduz o impacto do ataque no sistema; Há também um módulo de visualização que pode ser utilizado pelo administrador para ver o estado da rede. Em testes, o ML-IDS apresentou bons resultados quanto a falsos positivos. Atualmente o ML-IDS não pode detectar ataques que requerem identificação de carga útil do pacote e ataques a *hosts*, por que só monitora fluxos de rede.

Li *et al.* (2008) propõem o modelo $AHIT^{AC}$, que possui módulos de avaliação de confiança de acesso, ativação de armadilha para informações suspeitas, aprendizado de aplicações e recuperação de funções do sistema. Os autores afirmam que a primeira geração de tecnologia de segurança da informação é baseada em prevenção, a segunda em detecção, mas que naturalmente não é possível detectar todos os ataques e assim surge a terceira que é baseada em tolerância a intrusão. Desta forma, um sistema tolerante a intrusão deve garantir confidencialidade e integridade mesmo que o sistema esteja sofrendo ataques. Em outro trabalho (Li *et al.*, 2010), os autores apresentam o arcabouço ITAC (*Intrusion Tolerance Based on Autonomic Computing*) que monitora transmissões de rede, recursos internos/externos e os estados do ambiente para auto-avaliar o grau de credibilidade de um *host* e ativar armadilhas que transplantam os serviços suspeitos para uma espécie de quarentena. O modelo ITAC é formado por quatro partes: avaliação de credibilidade, mecanismo de decepção, classificação de serviço e módulo de execução, formando um ciclo dinâmico. O ITAC utiliza uma máquina de estados para avaliar os serviços como normal ou anormal dependendo do estado em que o serviço se encontra. O ITAC deve manter os serviços básicos funcionando mesmo que ataques estejam ocorrendo, ou seja, deve ser tolerante a ataques.

Chen *et al.* (2013) introduzem uma abordagem chamada ASM (*Autonomic Security Management*) para estimar, detectar e identificar ataques de segurança em redes de computadores. Nessa abordagem são utilizados sensores para coletar informações a respeito da rede. Os dados coletados são utilizados por IDS. Um controlador multi-objetivo seleciona um método ideal para recuperar o sistema do problema baseando-se nas assinaturas dos ataques. O ASM fica alojado em roteadores e é composto pelos módulos de previsão, sensores, processamento de dados, HIDS, NIDS, análise em tempo-real e o MAC (*Multi-objective Analysis Controller*). O MAC utiliza lógica *fuzzy* para decidir qual mecanismo de proteção utilizar frente a cada ataque. Os métodos de proteção utilizados são: IPS, que protege servidores Web de vários tipos de ataques examinando assinatura de pacotes; Filtro de pacotes; Plataforma confiável, que assegura que somente usuários legítimos acessem recursos do sistema quando o sistema está sob ataque; Replicação de servidor, que replica o servidor quando ele está sob ataque; Desconexão de rede, que desabilita placas de rede para bloquear o ataque, mas que pode levar ao bloqueio de clientes legítimos; Desligamento de *host*, que é usado caso um *host* comprometido não possa ser protegido pelos métodos citados anteriormente. A abordagem foi testada com os ataques DoS, injeção de SQL e exaustão de memória.

O trabalho descrito anteriormente (Chen *et al.*, 2013), é continuado em Chen *et al.* (2014), no qual é apresentado o ACSM (*Autonomic Cyber Security Management*) para estimar, detectar e reagir prematuramente ciberataques contra ambiente de Internet das Coisas – do inglês IoT (Internet *of Things*). Caso sejam detectados problemas, métodos tal como lógica *fuzzy* são empregados para determinar as melhores reações para mitigar problemas. As reações previstas são bloquear pacotes de rede ou substituir *hosts* comprometidos, que são aplicadas por meio de controladores de máquinas virtuais e por agentes presentes em servidores Web. Tanto Chen *et al.* (2013) quanto Chen *et al.* (2014), têm como foco principal, a escolha da reação autonômica. Já a proposta desta tese, visa analisar históricos de alertas de segurança e dados de uso da rede, para então gerar regras de segurança que mitiguem ciberameaças em redes de computadores locais. Também, a proposta de Chen *et al.* (2014) e Chen *et al.* (2013) utilizam agentes instalados em servidores Web, para servirem como sensores e atuadores, enquanto a presente tese emprega OpenFlow e NIDS.

Chu *et al.* (2010) propuseram um método autonômico para mitigar ataques (DDoS) por meio da contagem de volume de tráfego em *switches* OpenFlow. Desta forma, é caracterizado um DDoS observando-se a quantidade de pacotes transmitidos e pacotes enviados por segundo. Quando um limite pré-estabelecido é ultrapassado o ataque DDoS é confirmado e então o controlador OpenFlow começa a bloquear os pacotes relacionados com esses fluxos. Chu *et al.* (2010), também propõem usar LISP (*Location/ID Separation Protocol*) para criar uma base de dados de redes confiáveis. Assim, quando forem identificados ataques, os pacotes das redes identificadas como confiáveis poderão trafegar normalmente pelos *switches* OpenFlow, já os pacotes das redes desconhecidas serão descartados ou enviados para investigação. Em Chu *et al.* (2010), os autores não abordam como determinar quando um Id LISP é considerado confiável ou não. Comparado a proposta de Chu *et al.* (2010) com o da presente tese, esses têm como pontos similares: (i) utilização de contagem de volume de tráfego em *switches* OpenFlow para ajudar na identificação de ataques DDoS; (ii) a utilização de elementos OpenFlow para mitigar fluxos de redes maliciosos; (iii) a identificação de fluxos confiáveis. Contudo, a metodologia proposta nesta tese se difere por: (a) utilizar outros sensores, tal como IDS e não só os dados provindos dos *switches* OpenFlow para identificar ciberameaças, o que permite identificar outros tipos de ataques, não só DDoS; (b) empregar técnicas de aprendizado de máquina para analisar fluxos de redes maliciosos e legítimos, para então elaborar regras de segurança que mitiguem vários tipos de ataques contra a segurança da rede e beneficiar fluxos legítimos.

Apesar desta tese utilizar RDS apenas como um instrumento para criação/gerenciamento de sensores e atuadores, da arquitetura autonômica proposta, o Anexo A apresenta trabalhos que empregam RDS para mitigar ciberataques.

Em um contexto geral, os trabalhos apresentados neste capítulo têm apenas como ponto de intersecção com nosso trabalho o uso da CA na área de cibersegurança, ou seja, a metodologia empregada e até mesmo os objetivos são diferentes. Mesmo os trabalhos mais similares, que são os que abordam CA, cibersegurança e redes de computadores, se concentram na detecção de ciberameaças, enquanto o presente trabalho dedica-se mais na reação autonômica. Essa diferença de objetivos tem um lado positivo, pois demonstra que o presente trabalho contribui com uma área que não está sendo muito explorada/estudada na computação autonômica aplicada a redes de computadores (Barrère *et al.*, 2014). Todavia, a falta de trabalhos extremamente similares também dificulta a comparação entre resultados, já que a maioria relatam a respeito de resultados referentes a detecção e não dá eficácia em reagir contra ciberataques. Também dentre os trabalhos relacionados, há muitos que investigam resultados inerentes a *hosts* da rede (uso de CPU, memória, dentre outros) e não da influência dos ataques na rede como um todo, tal como é proposto nesta tese.

Por fim, o grande diferencial do presente trabalho, em relação aos outros, é que este se propõem a analisar os históricos de LAN e processá-los de forma similar a memória humana, utilizando para isso métodos de aprendizado de máquina não supervisionado. Para gerar regras de segurança que são aplicadas na rede por meio da tecnologia OpenFlow, para assim mitigar ciberameaças e proteger a rede. Além, disso a tese também inova ao propor métodos que permitem utilizar fontes de informações externas e não estruturadas, tal como o Twitter, para gerar alertas de segurança que informem o administrador da rede a respeito de problemas de cibersegurança que estejam afetando

as redes de computadores.

Capítulo 4

Metodologia utilizada para o desenvolvimento da arquitetura autonômica proposta

Objetivando melhorar o nível de segurança das redes de computadores e retirar parte do ônus do administrador em manter a segurança, principalmente das redes locais, este trabalho propõem uma arquitetura fundamentada em CA, que permite combinar tecnologias como RDS e IDS (Santos *et al.*, 2014).

O presente trabalho contribui com o estado da arte ao propor métodos que empregam algoritmos de análise de associação para processar históricos de uso da rede e alertas de segurança, extraindo desse processo regras de segurança que são aplicadas autonomicamente na rede, por intermédio de RDS, visando manter a normalidade da rede. Outra contribuição é a criação de um método inspirado na estrutura da memória humana, que permite (1) mitigar problemas de segurança na rede, (2) prevenir contra ameaças futuras e (3) evitar que *hosts* e/ou serviços da rede sejam influenciados por ataques e pela própria defesa autonômica, em caso de falsos positivos. Outra inovação proposta é a concepção de métodos que permitem extrair informações postadas em redes sociais para ajudar na identificação ou evidenciação, por parte da arquitetura ou do administrador da rede, a respeito de problemas de segurança que estejam afetando a rede ou que possam afetar futuramente (Santos *et al.*, 2016).

Assim, este capítulo apresenta a arquitetura proposta e os métodos empregados para possibilitar a reação autonômica frente a problemas de segurança em redes de computadores.

4.1 Arquitetura autonômica proposta

A arquitetura proposta é ilustrada pela Figura 4.1 sendo composta pelos módulos:

Sensores coleta dados a respeito dos diversos elementos da rede. Podem existir vários sensores em uma rede e esses devem fornecer informações referentes ao uso da rede e/ou alertas de segurança. São exemplos de sensores: NIDS e *switches* OpenFlow;

Monitor tem por objetivo normalizar e agregar as informações obtidas dos sensores bem como, ajudar na identificação de ameaças contra a segurança da rede. Conforme ilustrado na Figura 4.2, o módulo Monitor pode receber dados de vários sensores, que infelizmente podem seguir representações de dados diferentes. Desta forma, o submódulo Normalizador deve padronizar todas entradas de dados para que a arquitetura possa posteriormente processá-los. Os dados extraídos dos sensores são de dois tipos: (1) **dados a respeito do uso da rede**, que podem ser obtidos, por exemplo de sensores OpenFlow e (2) **alertas de segurança**, que podem ser gerados por sensores, tal como NIDS. O submódulo Analisador de Ameaças é responsável por analisar os dados de uso da rede em busca de anomalias, e caso sejam encontradas deve

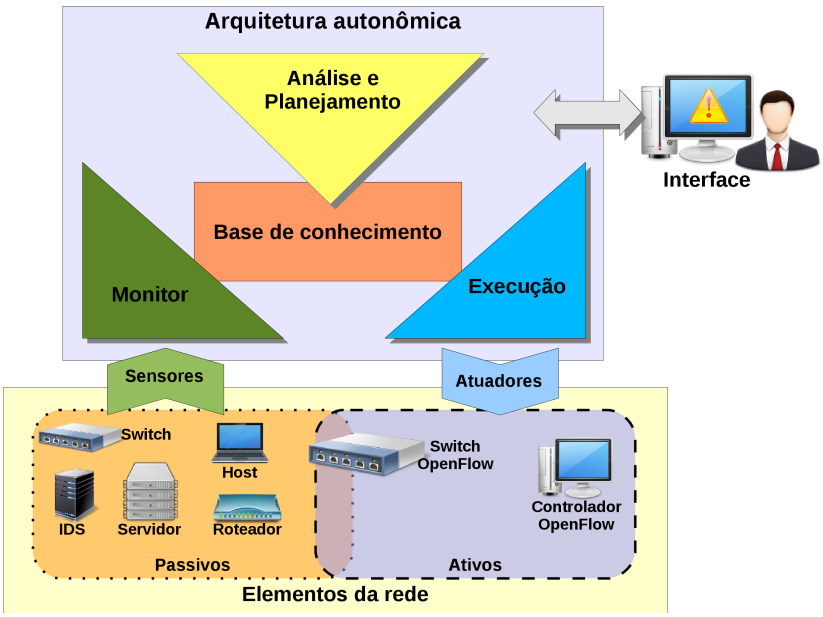


Figura 4.1: *Arquitetura autonômica proposta.*

gerar alertas de segurança, ou seja, esse submódulo complementa a identificação de ameaças feita por sensores externos (por exemplo IDS). Também, é função do submódulo Analisador de Ameaças relacionar os alertas de segurança gerados por sensores externos, com os dados de uso da rede. Os submódulos Dados de Uso da Rede e Alertas de Segurança permitem salvar dados respectivamente a respeito do uso da rede e alertas de segurança no módulo Base de Conhecimento e também recuperá-los de lá para que sejam processados pelo submódulo Analisador de Ameaças, que por sua vez pode ainda incluir/alterar dados na Base de Conhecimento, caso sejam evidenciados problemas de segurança;

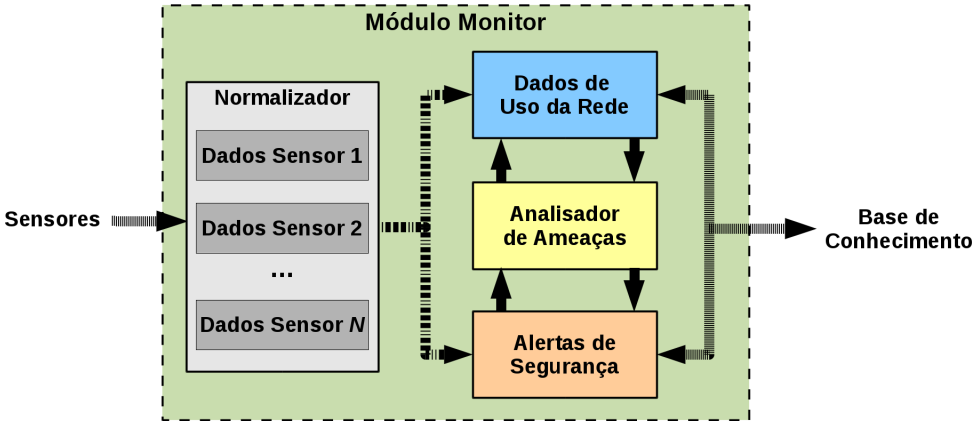


Figura 4.2: *Módulo Monitor.*

Base de Conhecimento recebe e armazena dados que servem de informações para tomada de ações por outros módulos da arquitetura. Esse módulo, que é ilustrado na Figura 4.3, trata três tipos de informações:

1. Registros locais - são os dados coletados pelo módulo Monitor, ou seja, são informações sobre o status da rede e alertas de segurança, o que é respectivamente representado na Figura 4.3 pelas bases de dados Alertas de Segurança e Dados de Uso da Rede no submódulo Registros Locais;
2. Políticas de segurança - são regras que ditam como os recursos da rede devem ser utili-

zados. A política de segurança pode ser redigida pelo administrador da rede ou criada autonomicamente pela própria arquitetura, o que diminui o trabalho do administrador e torna o ambiente mais autonômico. Isso é representado na Figura 4.3 pelo submódulo Política de Segurança que armazena as regras de segurança criadas pelo administrador na base de dados Regras de Segurança do Administrador. Já as regras de segurança criadas pela própria arquitetura são salvas em Regras de Segurança Autonômicas;

3. Vulnerabilidades Externas - são informações acerca de ameaças contra a segurança de computadores que são postadas em mensagens de redes sociais, tal como o Twitter. Esse submódulo tem por objetivo auxiliar na tomada de decisões a respeito da segurança da rede local, mas levando em consideração problemas externos, que estejam afetando a segurança de outras redes. A metodologia para se obter alertas de segurança de computadores a partir de mensagens postadas em redes sociais foi proposta e avaliada em Santos *et al.* (2012). Posteriormente o método foi melhorado e avaliado com uma base de dados maior no trabalho Campiolo *et al.* (2013). E em Santos *et al.* (2013), foi proposta e avaliada uma arquitetura que permite extrair alertas de segurança das mensagens postadas no *microblog* Twitter. Na Figura 4.3 o módulo Vulnerabilidades Externas representa toda a metodologia necessária para a extração de informações sobre vulnerabilidades e ameaças, postadas em redes sociais.

Então, o módulo Base de Conhecimento armazena históricos de conhecimento a respeito do ambiente local e de problemas externos, que servirão como alicerce para tomada de decisões;

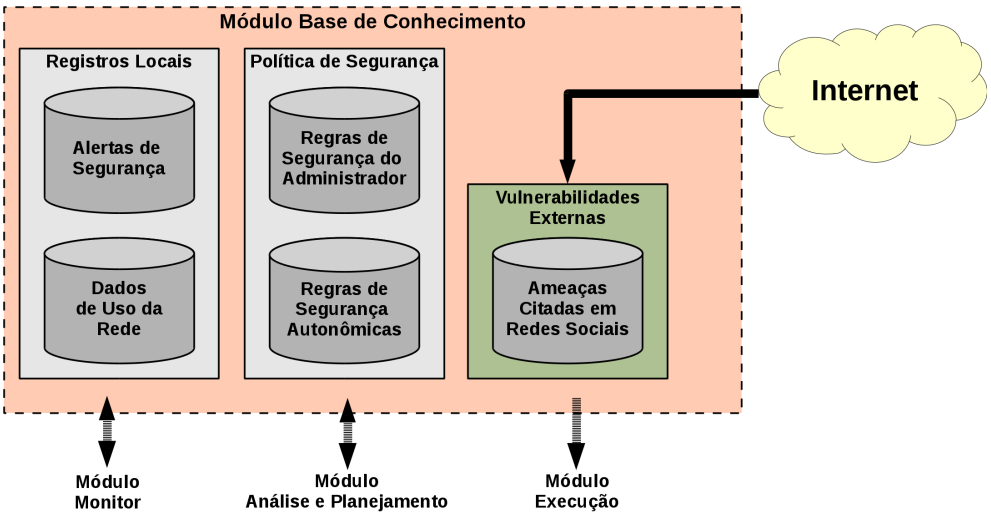


Figura 4.3: *Módulo Base de Conhecimento.*

Análise e Planejamento analisa os dados do módulo Base de Conhecimento em busca de sintomas que estejam influenciando o funcionamento da rede. Caso sejam evidenciados problemas, o módulo planeja ações para retomar o equilíbrio da rede. O processo de análise e planejamento emprega métodos, tal como aprendizado de máquina, para criar planos de ações que têm por objetivo manter ou restabelecer a normalidade da rede. O plano de ações escolhido deve levar em conta a política de segurança contida no módulo Base de Conhecimento. Assim, para manter a rede funcionando frente a ataques contra a segurança, o módulo Análise e Planejamento pode decidir por alterar a política de segurança criando, alterando ou apagando regras de segurança autonomicamente. Ou seja, a política de segurança pode ser criada e mantida pela própria arquitetura e para isso o módulo em questão deve atualizar a base de dados Regras de Segurança Autonômicas no módulo Base de Conhecimento. A análise de problemas de segurança na rede local pode ainda ser enriquecida através do relacionamento com informações contidas em redes sociais. Tal relacionamento é possível utilizando-se técnicas de

mineração de dados para cruzar e agrupar informações dos submódulos Registros Locais e Vulnerabilidades Externas do módulo Base de Conhecimento;

Execução cumpre o plano de ações armazenado no módulo Base de Conhecimento. O módulo Execução converte o plano de ações, criado para sanar problemas, em ações práticas que serão executadas em elementos da rede. Conforme pode ser visto na Figura 4.4 o módulo execução pode suportar vários atuadores. Então, a função do presente módulo é traduzir as regras de segurança que compõem a política de segurança em comandos a serem enviados para os atuadores. As regras de segurança que dão origem aos comandos são basicamente informações a respeito de *hosts*, serviços de rede e ações que devem ser aplicadas para manter o bom funcionamento da rede frente aos problemas de segurança. São exemplos de ações o bloqueio, restrições e redirecionamento de pacotes de rede;

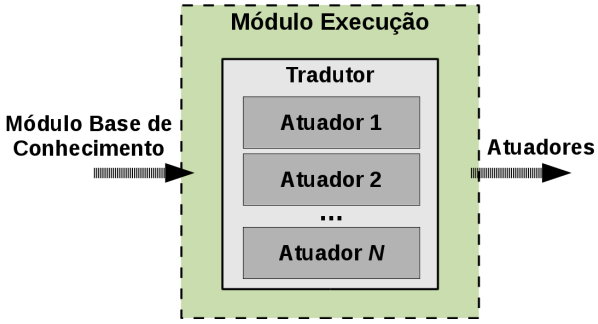


Figura 4.4: *Módulo Execução.*

Atuadores executa efetivamente as ações ordenadas pelo módulo Execução nos elementos da rede. Uma rede pode conter um ou mais atuadores. São exemplos de atuadores os *switches* OpenFlow e *firewalls*;

Elementos da rede são os diversos equipamentos que compõem a rede de computadores. A arquitetura proposta contempla dois tipos de elementos de rede, sendo estes: ativos e passivos. Os elementos passivos só podem enviar informações para os sensores e não interagem diretamente com os atuadores. Elementos de rede ativos são os mais completos, pois podem tanto enviar informações a respeito de seu estado por meio dos sensores, quanto executar comandos enviados pelos atuadores. A arquitetura proposta evita alterar as características básicas dos elementos de rede, devido a isso elementos como *hosts* clientes são normalmente passivos e portanto não são afetados diretamente pela arquitetura. Contudo os elementos passivos são indiretamente controlados pelas decisões autonômicas da arquitetura, pois estão ligados a elementos ativos, como *switches* OpenFlow e/ou *firewalls*, que irão afetar o comportamento de todos os elementos de rede;

Interface permite que o administrador configure a arquitetura autonômica e visualize o status da rede. Através da interface, o administrador da rede pode gerenciar manualmente a política de segurança incluindo, alterando ou excluindo regras de segurança na base de dados Regras de Segurança do Administrador do módulo Base de Conhecimento. Apesar da arquitetura ser autonômica é interessante permitir que o administrador redija regras de segurança que expressem seus anseios com relação ao uso da rede. Isso pode, por exemplo, inibir problemas com falsos positivos ou evitar ameaças à segurança previamente conhecidas pelo administrador. A interface também permite que o administrador visualize o status da rede, reações autonômicas da arquitetura frente a problemas de segurança e informações sobre problemas de segurança postadas na Internet. Isso possibilita que o administrador aumente seu conhecimento quanto ao que está acontecendo na rede local ou fora dela.

A arquitetura proposta e ilustrada na Figura 4.1 gera um ciclo autonômico, tal como a MAPE-K apresentada na seção 2.2.3.2. De forma prática, o ciclo autonômico proposto inicia com os sensores

coletando dados dos elementos passivos e ativos da rede: *hosts*, servidores, clientes, roteadores, *switches*, IDS, etc. Tais dados são repassados ao módulo Monitor que deve normalizá-los e auxiliar na identificação de alertas de segurança. Depois, esses dados são armazenados no módulo Base de Conhecimento, para que fiquem disponíveis aos demais módulos da arquitetura. O módulo Análise e Planejamento fica constantemente analisando o comportamento da rede, utilizando para isso os dados armazenados na Base de Conhecimento. Caso sejam identificados problemas na rede, o módulo planeja uma solução e disponibiliza o plano de ações para o módulo Execução. O módulo Execução irá traduzir as ações, para resolver o problema, em comandos que serão repassados aos atuadores. Os atuadores enviam esses comandos aos elementos ativos da rede: *switches*, *firewalls*, roteadores. Os comandos têm o intuito de mudar o comportamento dos elementos que compõem a rede e esse comportamento alterado será repassado a arquitetura pelos sensores, o que fecha o ciclo. A arquitetura deve observar se as ações tomadas resolveram ou não o problema, caso não, novas ações devem ser lançadas ciclo a ciclo, até que o equilíbrio seja restabelecido. O administrador da rede pode acompanhar e interagir com o ciclo autonômico através de uma interface. Caso o módulo Análise e Planejamento e/ou o administrador da rede fiquem em dúvida em identificar se um comportamento da rede é ou não um problema, esses podem recorrer a informações externas à rede, que estão disponíveis no módulo Base de Conhecimento por meio do submódulo Vulnerabilidades Externas.

Apesar da arquitetura autonômica proposta ter sido idealizada para funcionar em redes OpenFlow é possível utilizá-la com outras tecnologias. Para tanto basta integrar e adaptar novos sensores e atuadores, respectivamente aos módulos Monitor (Figura 4.2) e Execução (Figura 4.4).

4.2 Métodos e implementação

Como prova de conceito da arquitetura, foi desenvolvida uma aplicação autonômica denominada Of-IDPS, idealizada para zelar da segurança de ambientes de redes de computadores.

4.2.1 Visão geral do Of-IDPS

Como o nome sugere, o Of-IDPS, foi desenvolvido para ser executado em ambientes de rede que utilizam a tecnologia OpenFlow. O OpenFlow é a base da solução proposta, pois oferece facilidades na percepção do estado da rede e principalmente possibilita agir frente a imprevistos, alterando/programando a condição atual da rede, tal como necessita um sistema autonômico (ver Seção 2.2).

O Of-IDPS pode ser instalado em ambientes de rede, tal como o ilustrado na Figura 4.5. Idealmente, uma rede que utiliza a tecnologia OpenFlow possui dois segmentos: (1) um para a comunicação entre equipamentos OpenFlow (Canal de comunicação OpenFlow); e (2) outro para troca de informações entre elementos tradicionais (LAN comum). Todavia, é permitido agrupar tudo em uma única rede (Seção 2.3.1).

Observando a Figura 4.5 é possível notar que, a grosso modo, uma rede com Of-IDPS é composta por três elementos fundamentais:

***Switch* OpenFlow** *Switches* OpenFlow são tanto sensores quanto atuadores. A função de sensor é factível graças as mensagens OpenFlow de status da rede, tal como: *PortStatus*, *StatsReq*[1], que permitem obter informações a respeito do uso da rede. Os *switches* OpenFlow, também são ótimos atuadores, pois a tecnologia permite reprogramar a rede, e desta forma, criar, alterar e deletar fluxos de redes por intermédio de mensagens como: *PacketOut* e *FlowMod*[1]. Assim, é possível reduzir a largura de banda de fluxos suspeitos e impedir que fluxos maliciosos trafeguem nesses *switches*;

IDS O Of-IDPS também utiliza IDS de rede (NIDS) como fonte de informações a respeito de ameaças que estejam acometendo a rede, ou seja, sensores. A escolha do uso de IDS na arquitetura

[1] Ver Seção 2.3.1

se dá, pois essa já é uma tecnologia estudada, experimentada e consolidada, pela acadêmia e industria (ver Seção 2.1.2). Outro fator importante é que normalmente, empresas e comunidades que mantém os IDS se preocupam em deixá-los sempre atuais, frente as ameaças que surgem com o passar do tempo. Desse modo, o Of-IDPS empresta essas virtudes dos IDS para se manter na vanguarda da detecção de problemas. Inicialmente, foi considerado o uso de três NIDS de código aberto: Bro (Paxson, 2013), Suricata (OISFoundation, 2013) e Snort (Sourcefire, 2013), pois esses são muito conhecidos dentre os profissionais de cibersegurança. Porém, depois de experimentos preliminares e análises, decidiu-se usar em um primeiro momento, apenas o Snort devido a simplicidade de integração e por apresentar uma classificação de prioridade de alertas de segurança bem definida;

Controlador OpenFlow Em redes OpenFlow o controlador dita/programa como a rede deve se comportar (ver Seção 2.3.1). Justamente por isso, o núcleo do Of-IDPS fica alojado dentro do controlador OpenFlow. Dessa forma, o controlador OpenFlow aloca os módulos: Monitor; Base de Conhecimento; Análise e Planejamento; e Execução, da arquitetura autonômica proposta (Figura 4.1) e devido a isso, o termo Of-IDPS é sinônimo de controlador OpenFlow neste trabalho. Então, são as funções nativas do controlador OpenFlow somadas aos métodos da arquitetura proposta, que tornam possível a confecção de respostas autonômicas contra ameaças à segurança da rede. Os métodos que compõem o Of-IDPS e portanto o controlador OpenFlow, serão detalhados nas subseções seguintes.

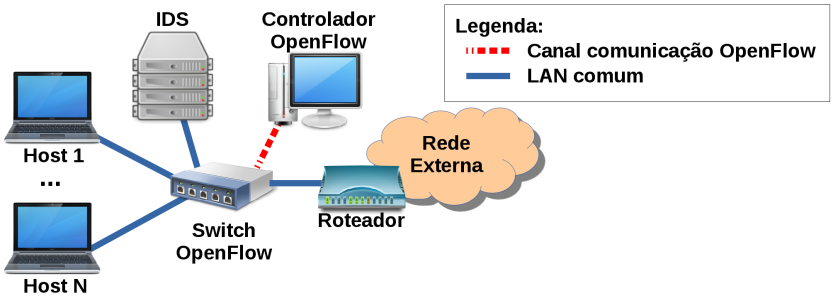


Figura 4.5: *Exemplo de cenário de rede Of-IDPS.*

Portanto, os elementos fundamentais do Of-IDPS dão suporte para o desenvolvimento da arquitetura autonômica proposta. A subseção a seguir descreve a organização prática do Of-IDPS em relação a arquitetura autonômica proposta.

4.2.1.1 Organização atual do Of-IDPS

A Figura 4.1, apresentada no início deste capítulo, ilustra a ideia geral da arquitetura proposta. Já a Figura 4.6 ilustra com mais detalhes a arquitetura proposta e algumas particularidades da sua organização para o desenvolvimento do Of-IDPS, principalmente dos métodos contidos no controlador OpenFlow.

Analisando a Figura 4.6 de maneira sucinta, é possível observar que o processamento proposto pela arquitetura, inicia com os sensores coletando informações a respeito do uso da rede e seus distúrbios (passo 1 - os passos são simbolizados pelos círculos com números na Figura 4.6). Na sequência, esses dados coletados são normalizados pelo módulo Monitor (passo 2), que também deve alertar a respeito de anomalias (passo 3). O histórico de uso da rede e os alertas de segurança são processados pelo módulo Análise e Planejamento (passo 4), que cria planos de ações para remediar problemas de segurança identificados e garantir o funcionamento de fluxos de redes legítimos. O módulo Execução converte o plano de ações em regras de segurança, que são implantadas na rede com dois propósitos: (i) mitigar fluxos de redes maliciosos que já estão instalados e ativos na rede (passo 5.1); (ii) repudiar ou restringir a comutação de novos fluxos, pelo controlador OpenFlow, quando esses estiverem relacionados com ataques à segurança (passo 5.2). O módulo Execução

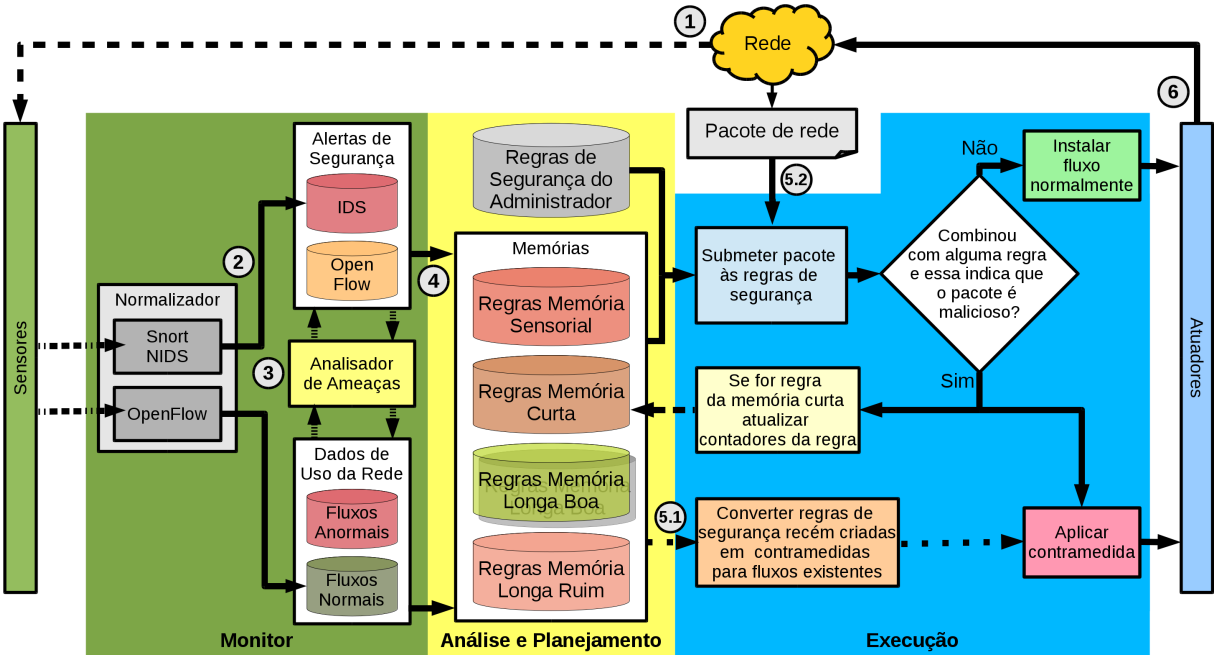


Figura 4.6: *Arquitetura detalhada do Of-IDPS.*

aplica as regras de segurança por intermédio dos atuadores (passo 6), assim fluxos honestos são instalados normalmente na rede, já fluxos maliciosos ou suspeitos são submetidos à contramedidas que visão resolver desequilíbrios na rede. Por fim, sensores retroalimentam (passo 1) a arquitetura fechando o ciclo autonômico infinito, similarmente ao que foi explicado na Seção 4.1.

Todas as informações geradas pelo Of-IDPS são registradas no módulo Base de Conhecimento para auxiliar em tomadas de decisões futuras. Na Figura 4.6, a Base de Conhecimento está espalhada e representada pelos cilindros: IDS, Fluxos Normais, Regras Memória Sensorial, etc. A Figura 4.7 relaciona o módulo Base de Conhecimento, apresentado na Figura 4.3, com as bases de dados do Of-IDPS. Assim, a relação base de dados do módulo Base de Conhecimento com as bases de dados do Of-IDPS são:

- As bases de dados IDS e OpenFlow do Of-IDPS representam a base de dados Alertas de Segurança da arquitetura;
- As bases Fluxos Anormais e Fluxos Normais formam a base Dados de Uso da Rede da arquitetura;
- A base Regras de Segurança do Administrador tem no Of-IDPS uma base de dados de mesmo nome;
- A base Regras de Segurança Autonômicas é representada na prática pelas bases Regras Memória Sensorial, Curta, Longa Boa e Longa Ruim;
- O relacionamento da base de dados Ameaças Citadas em Redes Sociais da arquitetura proposta com o Of-IDPS será melhor abordado na Seção 4.2.5.

O restante deste capítulo irá detalhar todos os módulos, métodos, bases de dados e passos, que permitem que a arquitetura autonômica proposta funcione.

4.2.2 Monitoramento no Of-IDPS

Tal como proposto na arquitetura o módulo Monitor no Of-IDPS é responsável por monitorar, normalizar e centralizar as informações dos vários sensores que a arquitetura pode utilizar.

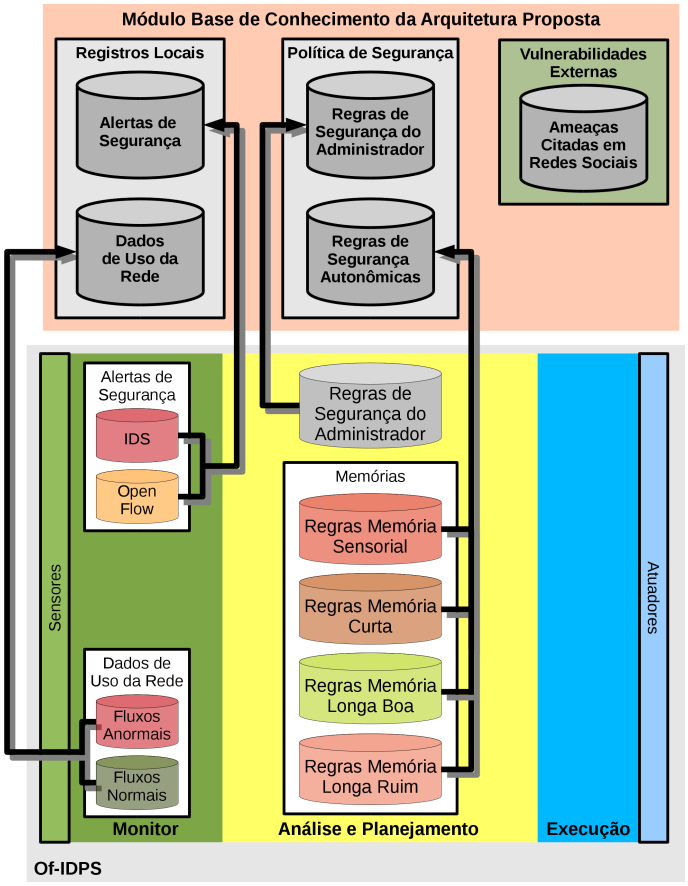


Figura 4.7: *Relação entre o módulo Base de Conhecimento e as bases de dados do Of-IDPS.*

Além disso, o módulo Monitor também pode detectar anomalias na rede, utilizando para esse fim estatísticas extraídas de mensagens de status da rede.

O Of-IDPS, representado pelo controlador OpenFlow, implementa o módulo Monitor que é composto pelos seguintes submódulos (ver Figura 4.6):

- Normalizador: padroniza o formato dos dados enviados pelos sensores. Atualmente o Of-IDPS utiliza o sensor Snort NIDS para monitorar incidentes de segurança na rede e o sensor OpenFlow para coletar dados a respeito do uso da rede. Caso, futuramente sejam adicionados mais sensores, tal como foi previsto na arquitetura proposta na Seção 4.1, novos submódulos serão acoplados para receber e normalizar os dados desses sensores. No Of-IDPS, a normalização exige basicamente dois passos a (1) extração de dados comuns entre os sensores (2) e padronização de tipos de dados. Esses passos serão melhor detalhados nas Subseções 4.2.2.0.1 e 4.2.2.0.2, respectivamente;

- Dados de Uso da Rede: submódulo responsável por armazenar as informações a respeito do uso da rede, que no caso do Of-IDPS são os dados provindos dos sensores OpenFlow. Os dados normalizados são mantidos em duas bases de dados: Fluxos Normais e Fluxos Anormais. A base Fluxos Normais armazena fluxos de rede considerados normais/comuns, ou seja, que não estejam relacionados com incidentes de segurança. A principio, todo fluxo de rede identificado e coletado pelos sensores são considerados honestos e por isso são salvos na base Fluxos Normais (passo 2 Figura 4.6). Já a base de Fluxos Anormais, armazena fluxos de rede que estejam relacionados com algum tipo de alerta de segurança, ou seja, que estejam em desacordo com a legalidade/normalidade da rede;

- Analisador de Ameaças: Emprega métodos para detectar anomalias na rede. A detecção de anomalias tem como fonte de informação as mensagens OpenFlow contidas na base Fluxos

Normais do submódulo Dados de Uso da Rede. Quando identificadas anomalias na rede, o submódulo deve gerar alertas e armazená-los na base de dados OpenFlow do submódulo Alertas de Segurança. Outra função do submódulo Analisador de Ameaças é trabalhar em conjunto com o submódulos Dados de Uso da Rede e Alertas de Segurança (flechas tracejadas interligando o submódulo Analisador de Ameaças aos submódulos Dados de Uso da Rede e Alertas de Segurança na Figura 4.6) para migrar fluxos relacionados problemas de segurança da base de dados Fluxos Normais para a base Fluxos Anormais. Ou seja, quando as informações a respeito de fluxos normais de rede combinam com um ou mais alertas de segurança, esses fluxos deixam de ter o status de normal e passam a ser considerados anormais;

- Alertas de Segurança: Responsável por armazenar alertas de segurança, que são gerados por sensores IDS e pelo submódulo Analisador de Ameaças. Assim, os dados desse submódulo são armazenados em duas bases: IDS e OpenFlow. A base de dados OpenFlow armazena os alertas de segurança gerados durante a detecção de anomalias efetuada pelo submódulo Analisador de Ameaças (passo 3 da Figura 4.6). A base de dados IDS, armazena os alertas de segurança gerados externamente pelos IDS.

A seguir as Subseções 4.2.2.0.1 e 4.2.2.0.1 irão discorrer em mais detalhes a respeito extração e normalização das informações provindas dos sensores. Já Subseção 4.2.2.1 descreve o processo de detecção de anomalias a partir da análise de fluxos OpenFlow.

4.2.2.0.1 Extração de dados

Devido ao fato do Of-IDPS ter por objetivo proteger redes de computadores, torna-se crucial obter informações pertinentes a transmissões de dados/pacotes na rede. Assim, no passo extração de informações vindas de sensores, do submódulo Normalizador tentá-se extrair, pelo menos os seguintes campos:

- Tempo: Representa a data e horário que a informação foi criada;
- Fonte: Identifica de onde veio a informação, pois essa pode ser originada de *switches* OpenFlow, IDS, etc;
- Descrição: Identificação ou breve explicação a respeito da informação. Por exemplo: Caso seja proveniente do OpenFlow, pode indicar que a informação é referente a um dado fluxo de rede; Já no caso da informação ser um alerta do NIDS Snort a descrição pode trazer um texto relatando o problema de segurança ou número/código que simboliza o problema;
- Prioridade: Representa o grau de importância/risco da informação. No caso de IDS alertas com prioridade alta significam grande risco à segurança. Já alertas com baixa prioridade representam menos risco. Caso algum sensor não trabalhe com prioridades, como é a situação das mensagens OpenFlow, o módulo Monitor irá deixar esse campo com a menor prioridade possível (0 - zero), simbolizando que não há prioridade. Ou seja, o Of-IDPS segue o princípio de quanto maior a prioridade mais riscos o problema relacionado com essa traz para a rede. Apesar do Of-IDPS utilizar números, no decorrer deste texto será utilizada a seguinte padronização de prioridades (para facilitar a compreensão):
  - *NÃO_HÁ*: Quando não há problemas de segurança;
  - *BAIXA*: Quando representa baixo risco a segurança do sistema;
  - *MÉDIA*: Quando representa risco médio a segurança do sistema;
  - *ALTA*: Quando representa grande risco a segurança do sistema;

  Essa nomenclatura de prioridade de segurança será usada em alertas e regras de segurança, bem como, em ações a serem tomadas em contramedidas de segurança;

- IP de origem: Endereço IP de origem. Considerando alertas de segurança, essa informação representa o *host* que está originando o ataque;
- IP de destino: Endereço IP de destino. Novamente no exemplo de alertas de segurança, essa informação representa o *host* que está sendo atacado;
- Protocolo: Informação contida na camada de rede e na prática é extraída do campo *Protocol* do cabeçalho do datagrama IP (Tanenbaum e Wetherall, 2013), isto é, indica o protocolo usado na transmissão. Comumente esse campo recebe os valores: TCP, UDP ou ICMP;
- Porta de origem: Representá o número da porta de transporte de origem, caso sejam utilizados os protocolos TCP ou UDP. Contudo, se for usado o protocolo ICMP, esse campo conterá o valor do campo *Type* do próprio protocolo ICMP (Tanenbaum e Wetherall, 2013);
- Porta de destino: Representá o número da porta de transporte de destino, caso sejam utilizados os protocolos TCP ou UDP. Contudo, se for usado o protocolo ICMP, esse campo conterá o valor do campo *Code* do próprio protocolo ICMP.

4.2.2.0.2 Padronização dos dados

Ainda no que se refere a normalização, após extrair os campos citados na subseção anterior, também é necessário padronizar seus respectivos tipos de dados.

A padronização se faz necessária, pois o Of-IDPS pode utilizar informações de diferentes tipos de sensores. Infelizmente cada sensor normalmente utiliza um formato de dados distinto. Logo, o módulo Monitor tem que convertê-los para um mesmo formato para poder processar corretamente esses dados.

Por exemplo, o IDS Snort - usando o Barnyard2 (Barnyard2, 2015) e banco de dados PostGreSQL (PostGreSQL, 2015) - representa endereços IP em um formato conhecido como *bigint* de 8 *bytes*, já o controlador OpenFlow, que usa a API (*Application Programming Interface*) Beacon (Erickson, 2014) trabalha com endereços IP no formato de números inteiros de 4 *bytes*. Assim, para que seja possível processar corretamente endereços IP, ou qualquer outro campo, é necessário convertê-los para que tenham o mesmo tipo de dados. Mais especificamente os dados dos sensores são convertidos para serem compatíveis com a API do Beacon, pois grande parte do Of-IDPS é desenvolvido tendo como base essa API.

Depois de todos os campos serem extraídos e padronizados, esses são enviados do submódulo Normalizador para suas respectivas bases de dados nos submódulos Alertas de Segurança e Dados de Uso da Rede para serem utilizados pela arquitetura autonômica (passo 2 da Figura 4.6).

4.2.2.1 Detecção de anomalias e problemas de segurança

O módulo Monitor também tem sob sua responsabilidade a detecção de problemas ou anomalias que podem afetar o bom funcionamento da rede.

De fato, as informações oriundas de IDS não necessitam passar pelo processo de detecção novamente, pois já são fruto de um mecanismo externo de detecção de ameaças, que é o próprio IDS. Desta maneira, os alertas gerados por IDS podem ser apenas normalizados e salvos diretamente na base de dados IDS do submódulo Alertas de Segurança (Figura 4.6).

Complementarmente aos alertas de IDS, o Of-IDPS detecta anomalias na rede através da análise de mensagens OpenFlow a respeito de estatísticas da rede, tal tarefa é representada pelo módulo Analisador de Ameaças da Figura 4.6, que é melhor detalhado na Figura 4.8.

O processo de detecção de anomalias, ilustrado na Figura 4.8, inicia com uma função que é responsável por obter alertas de segurança. Que por sua vez, dá inicio a execução de um ou mais métodos para evidenciar anomalias na rede por intermédio da análise dos dados contidos na base de dados Fluxos Normais do submódulo Dados de Uso da Rede. Caso os métodos não identifiquem anomalias e portanto não retornem alertas de segurança, nada será feito. Todavia, caso sejam

retornados alertas, a função que iniciou o processo insere os mesmos na base de dados OpenFlow do submódulo Alertas de Segurança e também move os fluxos de rede, antes considerados normais, para a base de dados Fluxos Anormais.

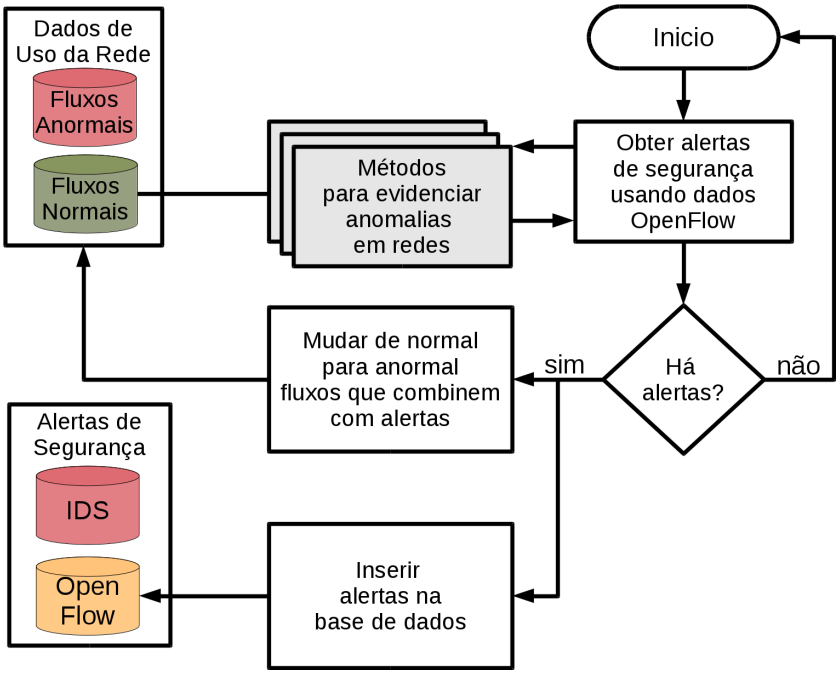


Figura 4.8: *Detecção de anomalias a partir da análise de dados OpenFlow.*

Atualmente, o Of-IDPS possui implementado um único método que ajuda na evidenciação de anomalias a partir da análise de mensagens OpenFlow. Tal método é representado essencialmente pela função *verificarOcorrenciaDDoS* (Algoritmo 1). A função recebe esse nome devido a deficiência verificada nos IDS, em especial o Snort, em realizar a detecção de ameaças de DDoS.

Portanto, o Algoritmo 1 verifica se há conexões suspeitas de DDoS em redes locais que utilizam a tecnologia OpenFlow. A constatação de DDoS é realizada mediante a contagem do número de pacotes transmitidos em cada fluxo de rede ativo nos *switches* OpenFlow. Caso haja 5 ou menos pacotes enviados em um dado fluxo, esse passa a ser considerado suspeito (linhas 4-6, do Algoritmo 1). Após contabilizar a quantidade de pacotes transmitidos em cada fluxo, o algoritmo pode retornar/gerar quatro tipos de resultados (linhas 8-19):

- NÃO_HÁ: Quando o número de fluxos ativos e suspeitos for menor ou igual a 20, não serão emitidos alertas de segurança (linhas 8 e 9). Assim, o termo "NÃO_HÁ" do algoritmo, significa que essa situação não representa riscos para a rede e não há alertas de DDoS;
- BAIXA: Quando o número de fluxos suspeitos for entre 20 e 50, serão emitidos alertas com prioridade de segurança baixa (linhas 10-12). Então, os alertas gerados aqui, representam pouca ameaça à segurança da rede. Será emitido um alerta de segurança para cada fluxo de rede suspeito (o mesmo se aplica para os itens seguintes);
- MÉDIA: Quando o número de fluxos suspeitos for entre 50 e 100, serão emitidos alertas de segurança com prioridade média (linha 13-15). São alertas que representam risco intermediário, contra as atividades da rede;
- ALTA: Quando o número de fluxos suspeitos for maior que 100, serão emitidos alertas de segurança com prioridade de segurança alta, ou seja, a rede provavelmente está correndo grande perigo (linhas 16-18).

A seleção do número de fluxos e pacotes para definir uma conexão suspeita considerou experimentos realizados com ferramentas de DDoS e comparações com cenários de tráfego reais e

Algoritmo 1: verificarOcorrenciaDDoS()

```
   Entrada: Informações a respeito de fluxos de rede OpenFlow
   Saída: Alertas de segurança extraído da análise de fluxos OpenFlow
 1 listaTodosFluxosAtivos ← obterFluxosAtivosNosSwitches()
 2 listaFluxosSuspeitos ← ϕ
 3 alertasDeSegurança ← ϕ
 4 para cada fluxo em listaTodosFluxosAtivos faça
 5     se fluxo.numeroPacotesNoFluxo ≤ 5 então
 6         listaFluxosSuspeitos.adiciona(fluxo)
 7 quantidadeFluxosSuspeitos ← listaFluxosSuspeitos.quantidade()
 8 se quantidadeFluxosSuspeitos ≤ 20 então
 9     alertasDeSegurança.atualizaPrioridade(NÃO_HÁ)
10 senão se quantidadeFluxosSuspeitos > 20 e quantidadeFluxosSuspeitos ≤ 50 então
11     alertasDeSegurança.converteFluxosEmAlertas(listaFluxosSuspeitos)
12     alertasDeSegurança.atualizaPrioridade(BAIXA)
13 senão se quantidadeFluxosSuspeitos > 50 e quantidadeFluxosSuspeitos ≤ 100 então
14     alertasDeSegurança.converteFluxosEmAlertas(listaFluxosSuspeitos)
15     alertasDeSegurança.atualizaPrioridade(MÉDIA)
16 senão
17     alertasDeSegurança.converteFluxosEmAlertas(listaFluxosSuspeitos)
18     alertasDeSegurança.atualizaPrioridade(ALTA)
19 retorna alertasDeSegurança;
```

simulados. As ferramentas utilizadas para analisar o comportamento de ataques DDoS foram ferramentas como: Nemesis[2]; DDoSim[3] e Hyenae[4]. O fluxo normal foi analisado de capturas de fluxos de redes obtidos durante uma pesquisa realizada pelos autores envolvendo alunos da UTFPR (Universidade Tecnológica Federal do Paraná) e com ferramentas de stress e teste de rede como o ab[5] e o Iperf[6]. Observou-se que ferramentas de ataques DDoS criam fluxos com menos de 5 pacotes e fluxos normais possuem um número superior. Logo, o limiar entre uma conexão normal e uma conexão suspeita foi estabelecido em 5 pacotes. Por fim, algumas conexões com poucos pacotes podem existir, mas dezenas se tornam perigosas e devem ser controladas. As suspeitas de DDoS também são reforçadas nas condições apresentadas anteriormente devido ao fato do OpenFlow remover automaticamente fluxos que não são usados por mais de 5 segundos. Assim, para que um alerta de DDoS seja emitido, os fluxos devem ser instalados em um intervalo mínimo de 5 segundos. Portanto, caracteriza-se como anomalia a ocorrência de 100 conexões ou mais, com apenas 5 pacotes cada, em um intervalo menor ou igual a 5 segundos. Ressalta-se que o objetivo é avaliar a detecção e reação autonômica do Of-IDPS e não propor um algoritmo de detecção de ataques DoS.

Desta forma, o módulo Monitor, após executar os métodos apresentados anteriormente, manterá um histórico com informações passadas e recentes que indicam a normalidade (base de dados Fluxos Normais) e os problemas da rede (bases de dados Fluxos Anormais, OpenFlow e IDS). Todas as bases de dados do módulo Monitor ajudam a compor o módulo Base de Conhecimento (Figura 4.1) e portanto servirão como fonte de informações para o restante da arquitetura, principalmente para o módulo Análise e Planejamento descrito a seguir.

4.2.3 Análise e planejamento no Of-IDPS

O módulo de Análise e Panejamento (Figura 4.1) tem por objetivo evidenciar e principalmente mitigar os problemas de segurança identificados pelo módulo Monitor.

Para que o Of-IDPS possa aprender com os ataques e problemas de segurança identificados pelo módulo Monitor, foi empregado aprendizado de máquina não supervisionado por meio do algoritmo FP-Growth (Han *et al.*, 2004), que é melhor detalhado na subseção seguinte.

[2] http://nemesis.sourceforge.net/
[3] http://sourceforge.net/projects/ddosim/
[4] http://sourceforge.net/projects/hyenae/
[5] http://httpd.apache.org/docs/2.2/programs/ab.html
[6] https://iperf.fr/

4.2.3.1 Conceitos a respeito de conjuntos de itens frequentes

O FP-Growth, utilizado neste trabalho, é similar ao histórico Apriori (Agrawal e Srikant, 1994), que permite descobrir/evidenciar relacionamentos entre informações em grandes volumes de dados. Essa técnica é conhecida como análise de associação ou regras de associação, e uma das abordagens é minerar os conjuntos de itens frequentes (*frequent itemsets*) (Harrington, 2012).

Então, algoritmos como FP-Growth e Apriori têm como entrada um conjunto de itens referentes a transações de banco de dados e como saída os itens que mais aparecem juntos nessas transações. Além do relacionamento entre os itens, outra informação resultante chama-se *suporte*, que nada mais é do que um valor numérico que representa a frequência/porcentagem de ocorrência de um conjunto de itens (Agrawal e Srikant, 1994; Han *et al.*, 2004).

Na prática tais algoritmos podem, por exemplo, ser utilizados para descobrir quais produtos são comumente comprados juntos em um supermercado, então, de posse dessas informações é possível desenvolver estratégias de mercado para obter mais lucros com a venda desses produtos (Harrington, 2012).

A Figura 4.9 ilustra a ideia básica do processamento de algoritmos para extrair conjuntos de itens frequentes. Nesse exemplo é extraída a relação de frequência de quatro itens/elementos: a, b, c e d. Para uma melhor compreensão, a explicação da figura foi dividida em quatro fases, sendo:

- Fase 1: Nesta é criado um conjunto (circulo) distinto para cada elemento, ou seja, são criados quatro conjuntos. Na sequência, é utilizado o suporte para determinar se um conjunto é frequente ou não. Neste exemplo, será considerado frequente conjuntos que apresentem suporte igual ou maior que 60%. É válido mencionar que o suporte é determinado por quem está executando o processamento. Na fase 1, o suporte representa o número de vezes que um dado elemento aparece sozinho nas transações analisadas, já que cada conjunto só tem um único elemento. No caso da Figura 4.9 o suporte está representado por uma porcentagem logo a baixo do conjunto. Logo, na primeira fase todos os conjuntos são considerados frequentes, pois possuem suporte entre 80% e 90%;

- Fase 2: São formados conjuntos com dois elementos/itens cada, então é calculado o número de vezes que os dois itens combinados, de cada conjunto, aparecem juntos nas transações processadas. Nesse caso há um único conjunto que apresenta suporte inferior a 60%, assim o conjunto cd é considerado infrequente, ou seja, os itens c e d só aparecem juntos em 30% das transações analisadas. Os outros conjuntos atingem o suporte desejado;

- Fase 3: Os elementos são relacionados em conjuntos de três. Nesta fase os conjuntos acd e bcd não atendem ao suporte e portanto são considerados infrequentes. Já os conjuntos abc e abd são frequentes;

- Fase 4: Por fim, é processado o suporte dos quatro elementos/itens juntos e essa relação não é considerada frequente. Assim, se os itens a, b, c e d simbolizassem produtos vendidos em um supermercado, significaria que tais itens são vendidos juntos em apenas 10% do total das compras realizadas pelos clientes.

Para processar conjuntos de itens frequentes com N elementos são necessários $2^N - 1$ conjuntos. Assim, no exemplo da figura 4.9 são utilizados 15 conjuntos (círculos), pois com 4 elementos têm-se $2^4 - 1$ conjuntos. Então, processar conjuntos de itens frequentes é um problema exponencial e justamente para reduzir o poder computacional necessário para processar esses conjuntos que surge um principio chamado de Apriori (Harrington, 2012), que afirma: "*se um conjunto é infrequente, então todos os seus superconjuntos também serão infrequentes*" (Agrawal e Srikant, 1994). Em outras palavras, se um conjunto não atender ao suporte exigido, então não é necessário processar os seus superconjuntos, pois esses também não atenderão o suporte mínimo exigido. O principio Apriori pode ser visto na Figura 4.9, pois na fase 2 o conjunto cd não atinge o suporte desejado e nas fases seguintes todos os superconjuntos derivados deste, também são considerados infrequentes - os conjuntos que atendem o principio Apriori foram delineados com linhas ultra-tracejadas na figura.

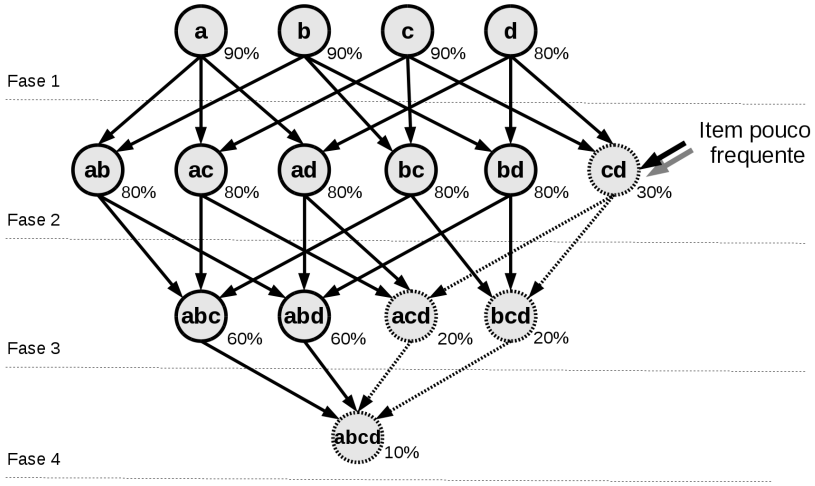


Figura 4.9: *Exemplo de conjuntos de itens frequentes (Harrington, 2012).*

Então, o algoritmo Apriori torna factível o processamento de conjuntos de itens frequentes. Contudo atualmente existem algoritmos superiores ao Apriori. O FP-Growth por exemplo, utiliza uma estrutura de dados conhecida como FP-tree para construir um padrão de frequência entre os itens que atendem ao suporte e devido a isso requer menos memória do computador e apresenta também um tempo de execução menor que o Apriori (Han *et al.*, 2004). Então, devido a melhor eficiência do FP-Growth relatada em trabalhos como Fournier-Viger (2014); Hunyadi (2011); Kosters *et al.* (2003), esse foi escolhido para ser empregado neste trabalho. O uso do FP-Growth no Of-IDPS é apresentado a seguir.

4.2.3.2 Aprendizado de máquina utilizado no Of-IDPS

No contexto deste trabalho, é proposta a submissão dos dados das bases IDS, OpenFlow e Fluxos Normais (passo 4 da Figura 4.6), como transações a serem analisadas pelo algoritmo FP-Growth, que então devolverá como como saída: (i) padrões de ataques, fundamentados nos alertas de segurança; e (ii) padrões de uso da rede, embasado em dados fornecidos pelo OpenFlow.

No Of-IDPS, cada alerta de segurança a ser submetido no FP-Growth é composto pelos campos endereço IP de origem, endereço IP de destino, protocolo (TCP, UDP, ICMP, etc), porta de origem, porta de destino, identificação/descrição do problema de segurança e o grau de risco/prioridade do problema de segurança que gerou o alerta, que são alguns dos campos normalizados pelo módulo Monitor, descritos na Subseção 4.2.2.0.1. A Tabela 4.1 ilustra alertas fictícios.

Os alertas, bem como o suporte mínimo esperado, são submetidos ao FP-Growth, que devolve conjuntos de itens que atendem ao suporte. Cada conjunto de itens obtido a partir dos alertas gerará uma regra de segurança específica, sendo que as regras de segurança possuem os mesmos campos dos alertas de segurança. Desta forma, para criar as regras basta pegar um conjunto retornado pelo FP-Growth e copiar os campos/itens para a regra de segurança. Todavia, em alguns casos, um conjunto pode não retornar todos os campos de um alerta por não atender ao suporte. Caso isso ocorra, os campos faltantes são preenchidos com um valor especial, tal como o simbolo *, que simboliza a combinação com qualquer informação de pacote de rede – abordagem similar com o que acontece na especificação de regras de *firewalls*.

Por exemplo, se as entradas da Tabela 4.1 forem submetidas ao FP-Growth com o suporte mínimo de 60%, o algoritmo devolverá cinco conjuntos de itens, que resultarão nas regras de segurança apresentadas na Tabela 4.2, que serão instaladas nos *switches* (passo 5.1 da Figura 4.6) e no controlador OpenFlow para conter os fluxos maliciosos (passo 5.2 da Figura 4.6). Todos os campos da Tabela 4.2 que estão preenchidos com o simbolo de *, são campos que não estavam presentes no conjunto de itens que criou a regra. Então, analisando apenas a linha 1 da Tabela 4.2, é possível notar que o conjunto de itens retornado, que formou essa regra, só possuía os campos

IP de origem, protocolo, porta de destino e prioridade do alerta de segurança e, todos esses campos combinados apareceram em pelo menos 60% (suporte mínimo) das transações da Tabela 4.1 (campos sublinhados). Já os campos IP de destino, porta de origem e identificação do alerta de segurança não apresentaram o suporte necessário quando combinados com os demais campos dessa regra. Assim, a regra apresentada na linha 1 da Tabela 4.2, por exemplo, especificará que pacotes originados do endereço IP 20.0.0.1, destinados a qualquer endereço IP, utilizando o protocolo de rede TCP, originados de qualquer porta, destinados a porta 22, serão afetados considerando o nível de prioridade de segurança Médio, o que pode levar a uma ação de bloqueio ou de restrição no fluxo de dados.

Tabela 4.1: *Exemplo de alertas a serem processados pelo FP-Growth.*

N°	Endereço IP		Protocolo	Portas de rede		Alerta de segurança	
	Origem	Destino		Origem	Destino	Identificação	Prioridade
1	20.0.0.1	30.0.0.10	TCP	50000	22	SSH01	Média
2	20.0.0.1	30.0.0.20	TCP	50001	22	SSH02	Média
3	20.0.0.1	30.0.0.30	TCP	50002	22	SSH03	Média
4	20.0.0.1	30.0.0.40	TCP	50003	22	SSH01	Média
5	20.0.0.1	30.0.0.50	TCP	50004	22	SSH01	Média
6	20.0.0.1	30.0.0.60	TCP	50005	22	SSH01	Média
7	20.0.0.2	30.0.0.70	UDP	50006	53	DNS01	Alta
8	20.0.0.2	30.0.0.80	UDP	50007	53	DNS01	Alta
9	20.0.0.2	30.0.0.90	UDP	50008	53	DNS01	Alta
10	20.0.0.1	30.0.0.10	ICMP	0	1	ICMP01	Baixa

Tabela 4.2: *Exemplo de regras criadas a partir dos alertas de segurança*

N°	Endereço IP		Protocolo	Portas de rede		Alerta de segurança		Suporte (%)
	Origem	Destino		Origem	Destino	Identificação	Prioridade	
1	20.0.0.1	*	TCP	*	22	*	Média	60
2	20.0.0.1	*	TCP	*	*	*	Média	60
3	20.0.0.1	*	*	*	*	*	Média	70
4	*	*	TCP	*	22	*	Média	60
5	*	*	TCP	*	*	*	Média	60

Contudo, analisando as regras de segurança da Tabela 4.2, constata-se que existem alguns problemas com as regras de segurança geradas pelo FP-Growth. Assim, esses problemas e as respectivas soluções (representadas no Algoritmo 2), são apresentadas a seguir:

- Há regras conflitantes. Por exemplo, quando o controlador OpenFlow analisar um pacote (passo 5.2 da Figura 4.6) originado do *host* 20.0.0.1 destinado a porta TCP 22, haverá ambiguidade em se descobrir qual regra aplicar, pois todas regras criadas combinam com esse pacote. Para resolver esse problema, aplicá-se a regra que combina com a maior quantidade de campos do pacote e representa maior ameaça ao sistema (verificado pela prioridade de segurança), por fim, caso o impasse continue, aplica-se a primeira regra dentre as restantes;

- Algumas combinações de campos não são úteis. Por exemplo, não é possível formar uma regra somente com os campos: prioridade de segurança e/ou identificação do problema. Também, regras contendo portas só são utilizadas se aparecerem acompanhadas do campo protocolo. Então, regras compostas somente com os campos de descrição de alerta, prioridade, portas ou combinando esses três campos são descartadas. A resolução desse problema é representada pelo método *removerRegrasQuePossuemApenasCamposInvalidosParaFormacaoDeRegras()* do Algoritmo 2;

- O uso do algoritmo FP-Growth original pode gerar regras muito genéricas, o que pode influenciar a rede inteira, incluindo fluxos não maliciosos. Um exemplo disso é o caso da regra 5 da Tabela 4.2, que aplica restrições para todos os fluxos TCP na rede. Para resolver esse problema propõem-se o uso de um suporte mínimo que varia em relação ao número de itens retornado pelo conjunto de itens (linhas 5-9 do Algoritmo 2). Desta forma, exige-se que conjuntos com poucos itens precisam apresentar um número de suporte maior do que conjuntos com muitos itens. A contagem dos itens para esse método desconsidera os campos identificação

e prioridade de segurança, já que as regras são aplicadas considerando apenas campos pertinentes à rede, ou seja, contabiliza-se apenas IP origem, IP destino, protocolo, porta origem e porta destino. Um caso especial é quando a regra tem apenas o campo protocolo, pois como esse tipo de regra é muito abrangente, então também são descartadas no método *removerRegrasQuePossuemApenasCamposInvalidosParaFormacaoDeRegras()* do Algoritmo 2. Assim, submetendo os alertas da Tabela 4.1 ao Algoritmo 2, a saída será apenas a regra 1 da Tabela 4.2, pois essa é a única que atende os requisitos exigidos.

Algoritmo 2: gerarRegrasDeSegurançaAutonômicas()

```
   Entrada: listaAlertasSegurança - Lista de alertas gerados pelo Of-IDPS
   Saída: Lista com regras de segurança
 1 listaConjuntoRegras ← φ
 2 listaRegrasSegurança ← φ
 3 listaConjuntoRegras ← processaAlertasComAlgoritmoFPGrowth(listaAlertasSegurança)
 4 listaConjuntoRegras ← removerRegrasQuePossuemApenasCamposInvalidosParaFormacaoDeRegras(listaConjuntoRegras)
 5 para cada regra em listaConjuntoRegras faça
 6     suporte ← regra.obterSuporte()
 7     numeroItens ← regra.obterNumeroItensPresenteNaRegra()
 8     percentualSuporte = 110 − 20 ∗ numeroItens
 9     se suporte ≥ percentualSuporte então
10         listaRegrasSegurança.adiciona(regra)
11 retorna listaRegrasSegurança
```

Portanto, o uso do Algoritmo 2 propicia a criação autonômica de regras de segurança no Of-IDPS. Entretanto, ainda há um grande problema, pois conforme o tempo for passando e os alertas de segurança forem surgindo, a base de dados de alertas crescerá gradativamente, o que pode causar um problema conhecido como *overfitting* Harrington (2012), durante a análise dos dados. Ou seja, com um grande número de alertas antigos na base de dados surge a possibilidade desses alertas, em algum momento, começarem a ofuscar alertas novos, o que influenciaria negativamente na criação de regras autonômicas para mitigar distúrbios de segurança recentes.

Por exemplo, a Figura 4.10 ilustra problemas de segurança que estão afetando ou aferaram uma dada rede. Na figura, cada problema gera um alerta de segurança que é representado por um retângulo identificado por uma letra. Também, há uma linha de tempo, que auxiliada pelas chaves (passado, recente e agora) dão noção de qual período de tempo cada alerta se encontra. Para facilitar o entendimento do problema será analisado/considerado apenas a identificação do alerta (letra do retângulo), ou seja, serão desconsiderados os detalhes e possíveis relacionamentos entre alertas, que foram sugeridos e explicados anteriormente nesta seção. Assim, analisando todo o histórico de ataques de maneira simplista, é possível verificar que o problema que mais ocorre é o y, representando 70% dos ataques da rede, seguido por: z com 15%; x com 10%; e w com 5%. Desta maneira, caso fosse utilizado um "suporte mínimo" de 60% – que é o mesmo valor de suporte empregado anteriormente no exemplo da Tabela 4.2 – seriam criadas apenas regras de segurança para conter o ataque y, pois nenhum outro tipo de ataque atinge o suporte mínimo. Todavia, considerado ataques atuais na rede (chave Recente), nota-se que seria interessante remediar também o ataque de segurança z, que vêm comprometendo a rede recentemente. Pior, segundo a Figura 4.10, neste exato momento a rede está sendo afetada pelo problema de segurança x (chave Agora). Contudo, infelizmente nem o ataque z nem o x serão mitigados, já que respectivamente apresentam um "suporte" de apenas 15% e 10%, assim não serão criadas regras de segurança para contê-los.

Então, processar conjuntamente todos os alertas (antigos e recentes) pode não ser uma boa ação. A solução para esse problema é apresentada na próxima subseção.

4.2.3.3 Reação autonômica inspirada na memória humana

Para resolver conflitos entre alertas antigos e recentes, ou seja, possibilitar que o Of-IDPS utilize informações de ataques já ocorridos no passado ou que estão ocorrendo no presente para prevenir novos ataques e também remediar problemas em andamento, foi desenvolvido um mecanismo baseado na estrutura da memória humana.

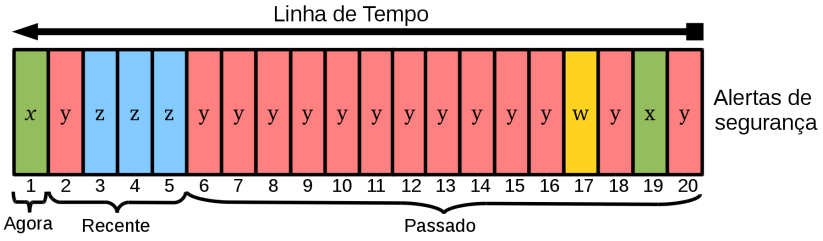


Figura 4.10: *Exemplo de base de dados de alertas de segurança identificados por cores e letras.*

Sucintamente, a memória humana é processada/dividida fundamentalmente em três memórias (Baddeley, 1997):

- **Sensorial**: Trata de informações que estão ocorrendo atualmente e está ligada aos sentidos do corpo humano, ou seja, é uma memória de curtíssima duração responsável por armazenar sensações (visual, tátil, etc) que estejam ocorrendo no presente;
- **Curta**: Armazena informações passadas, mas que ocorreram recentemente. Esta também é chamada de memória de curto-prazo;
- **Longa**: Armazena acontecimentos marcantes que ocorreram em um passado mais distante se comparado com a memória curta. Esta também é conhecida como memória de longo-prazo.

Então, a ideia é viabilizar ao Of-IDPS uma forma de processar o histórico da rede segundo a mesma política. Consequentemente, o Of-IDPS também implementa as memórias: sensorial, curta e longa, que serão discutidas respectivamente nas Subseções 4.2.3.3.1, 4.2.3.3.2 e 4.2.3.3.3, a seguir.

4.2.3.3.1 Memória Sensorial

A memória sensorial do Of-IDPS é responsável por processar os alertas de segurança que acabaram de ser identificados pela arquitetura, em outras palavras, alertas extremamente recentes. Pois, esses alertas representam o que a rede está "sentindo" naquele exato momento, tal como na memória humana. Então, os alertas de segurança processados pela memória sensorial dão origem a regras de segurança que são aplicadas imediatamente na rede visando sanar o problema que está ocorrendo no momento.

No Of-IDPS, a memória sensorial é alimentada pelos alertas de segurança recém identificados pelos sensores e armazenados nas bases de dados do submódulo Alertas de Segurança do módulo Monitor (Figura 4.6). Também, a memória sensorial possui uma base de dados chamada Regras Memória Sensorial, que é responsável por armazenar as regras de segurança criadas para mitigar os ataques evidenciados por esta memória.

Conforme pode ser observado na Figura 4.11, o processo de criação de regras de segurança pela memória sensorial, inicia com uma função que apaga todas as regras de segurança previamente armazenadas na base Regras Memória Sensorial. Na sequência, outra função obtém os alertas de segurança que estão afetando o sistema naquele momento, ou seja, são retornados apenas alertas de segurança de alguns poucos segundos atrás. A partir dai haverão duas possibilidades:

1. Não serão retornados alertas de segurança. Simboliza que a rede não está sendo atacada naquele exato momento. Nesse caso, não é necessário criar/ter regras de segurança na memória sensorial, já que não há perturbação presente na rede;
2. São retornados alertas de segurança. Significa que a rede está sofrendo ataques naquele momento. Então, é necessário criar regras de segurança para proteger a rede do ataque em andamento. Nesse caso, para cada alerta retornado, serão criadas duas regras de segurança para conter o ataque do referido alerta. Todas as regras geradas serão armazenadas na base

Regras Memória Sensorial e posteriormente serão aplicadas na rede pelo módulo Execução (descrito posteriormente).

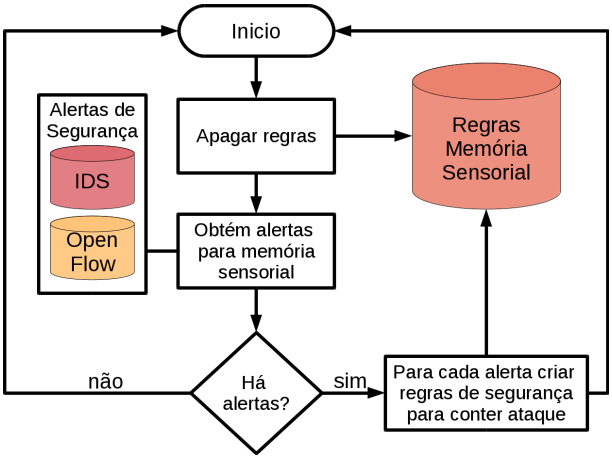


Figura 4.11: *Geração de regras de segurança na memória sensorial.*

No Of-IDPS a memória sensorial é a única que não emprega o algoritmo FP-Growth para gerar regras segurança, isso porque tal memória deve atacar o exato problema/alerta que está em curso. Assim, o método empregado é criar duas regras de segurança para cada alerta identificado. Cria-se duas regras por alerta para impedir que os pacotes de rede, relacionados com o referente ataque, fluam em ambas direções, ou seja, entrando e saindo da rede.

A Figura 4.12 mostra um exemplo de criação de regras de segurança de um dado alerta da memória sensorial. Basicamente o processo consiste em pegar todos os campos do alerta e transcrevê-los para os respectivos campos da regra de segurança 1, exceto pelo campo porta de origem (na regra 1 essa é a porta cliente) que é generalizado (simbolo *). A regra de segurança 1, trata a entrada de pacotes do atacante na rede, o próximo passo é criar a regra 2 para mitigar a resposta da vítima. Para criar a regra 2, basta transcrever os campos da regra 1, com o cuidado de inverter respectivamente os campos IP de origem e porta de origem com os campos IP de destino e porta de destino.

A generalização da porta cliente, citada anteriormente, é necessária já que essa porta é aleatória, ou seja, no próximo acesso do atacante para a vítima, a porta cliente será alterada, pois é um comportamento padrão da rede (Tanenbaum e Wetherall, 2013). Desta forma, simplesmente transcrever a porta cliente limita muito a capacidade da regra em mitigar o atacante, já que provavelmente esse em uma próxima tentativa de ataque irá acessar a rede usando outra porta cliente, nesse caso o Of-IDPS teria que criar outra regra de segurança para impedir o "novo" ataque, o que não é necessário caso se generalize a dita porta cliente. Essa técnica da generalização das portas do lado cliente é muito utilizada na criação de regras em *firewall*.

No Of-IDPS, o processo de analise de alertas de segurança e criação de regras ilustrado pela Figura 4.11 é executado de forma continua por meio de uma *thread* (Tanenbaum, 2007). Portanto, da mesma forma que na memória sensorial humana, tanto os alertas de segurança quanto as regras geradas, são todos criados e removidos bem rapidamente da memória sensorial do Of-IDPS.

4.2.3.3.2 Memória Curta

A exemplo da humana, a memória curta do Of-IDPS retém alertas por apenas alguns segundos. É nesta memória que começa a ser aplicado o Algoritmo 2, com o objetivo de aprender com os problemas ocorridos na rede recentemente e criando dessa maneira regras de segurança para mitigar ameaças que estejam afetando a rede ou que poderão afetar em um futuro próximo.

A Figura 4.13 ilustra o fluxo de processamento da memória curta do Of-IDPS. Todo o processo

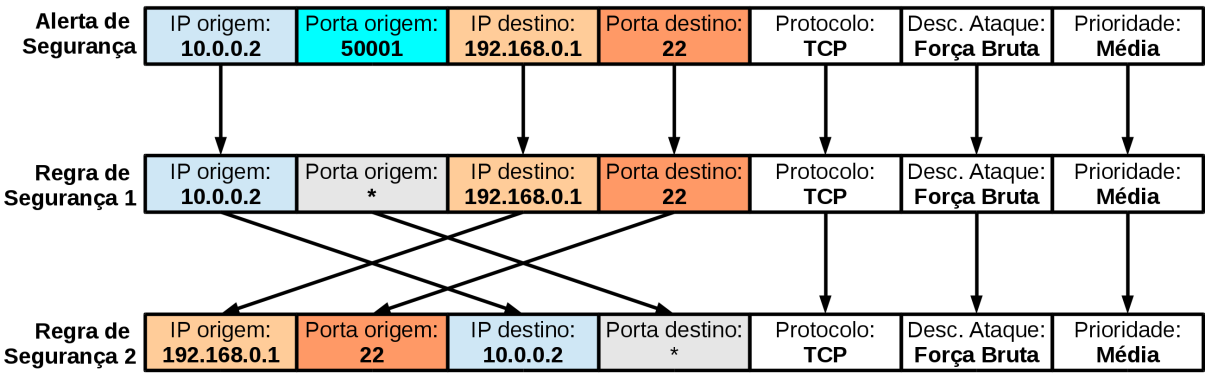


Figura 4.12: *Exemplo de criação de regras na memória sensorial.*

inicia com a obtenção de alertas de segurança ocorridos recentemente e que foram registrados pelo módulo Monitor. A partir desse ponto há duas possibilidades:

1. Caso sejam retornados alertas de segurança, esses são submetidos ao algoritmo FP-Growth, que irá gerar padrões de ataques fundamentados nos alertas. Na sequência são aplicados métodos empíricos para remover conjuntos de itens frequentes indesejados, tal como descrito anteriormente na Subseção 4.2.3.2. De posse das regras de segurança recém criadas pelos métodos propostos, essas devem ser mantidas em uma base de dados chamada Regras Memória Curta;

2. Caso não sejam retornados alertas de segurança, a memória curta deve apenas atualizar as regras de segurança preexistentes na base de dados Regras Memória Curta.

Um diferencial da memória curta em relação a outras memórias, é que esta possui um submódulo específico para gerenciar a inclusão, atualização e exclusão das regras de segurança em sua base de dados. Os motivos desse gerenciamento diferenciado é exposto a seguir.

Gerenciamento de regras da memória curta

Se compara com outras memórias do Of-IDPS, a memória curta utiliza uma política mais elaborada para manter (incluir, atualizar e excluir) as regras de segurança em sua base de dados. Isso se faz necessário, pois um problema enfrentado pela memória curta é que ao criar as regras de segurança, essas naturalmente podem/devem bloquear os fluxos maliciosos que estavam gerando os alertas. Todavia, com os fluxos maliciosos bloqueados não serão mais emitidos alertas de segurança. Sem alertas alimentando a memória curta o sistema pensará que não há risco eminente e destituirá as regras de segurança que policiavam o ataque, o que pode dar brecha ao ressurgimento do mesmo.

O problema descrito no paragrafo anterior pode fazer com que o ataque surja e desapareça constantemente na rede, como se fosse as subidas e descidas de uma montanha russa, que vem e vão sem parar. Então, são utilizados dois artifícios no Of-IDPS, para resolver o problema:

1. Toda regra de segurança na base de dados Regras Memória Curta tem um indicador de uso da regra, que é uma espécie de tempo de vida. Assim, quando a regra é criada, lhe é atribuída um tempo de vida que é decrementado, de tempos em tempos, caso a regra não seja utilizada. Quando o tempo de vida atingir um valor mínimo (0 – zero), a regra é removida do sistema. Contudo, caso a regra seja utilizada no combate a problemas na rede, tal como pelo módulo Execução (Seção 4.2.4), seu tempo de vida é incrementado (atualizado). Todavia, é instituído um valor teto que não pode ser ultrapassado, evitando desta maneira que a regra ganhe valores muito altos e permaneça instalada de forma desnecessária durante muito tempo;

2. O módulo "Manter regras de segurança" existe justamente para gerenciar as regras de segurança armazenadas na base Regras Memória Curta. Assim, quando uma regra chega neste

módulo, sua primeira função é verificar se essa é realmente nova, caso seja, a regra será incluída/adicionada na base de dados e seu tempo de vida será configurado com um valor padrão. Caso o módulo constate que a regra criada já existia na base de dados, essa terá o seu tempo de vida incrementado (atualizado), pois significa que ainda há alertas a respeito do referido ataque e portanto a regra é útil. Após processar as regras novas, o módulo decrementa o tempo de vida das regras restantes e verifica se alguma regra expirou, caso sim, essas são removidas da memória curta.

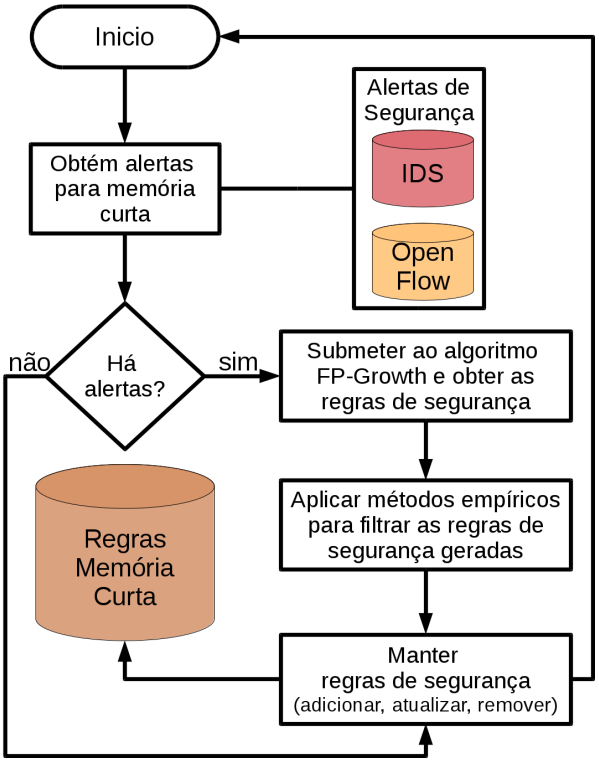


Figura 4.13: *Geração de regras de segurança na memória curta.*

Como mencionado, as memórias sensorial e longa, não possuem um gerenciamento de regras de segurança tão complexo quanto a memória curta. Isso se deve ao fato dessas memórias lidarem com os problemas de segurança em contextos diferentes. Ou seja, a memória sensorial deve tratar apenas de problemas imediatistas. Assim, tão logo um problema seja identificado pela arquitetura esse é primeiro e rapidamente tratado de forma pontual pela memória sensorial e assim que o referido empecilho seja sanado, as regras de segurança que o reprimiram devem ser removidas para que a rede volte ao seu estado normal. Já no contexto da memória longa, que será detalhada na subseção seguinte, as regras de segurança não precisam ter contadores para serem mantidas nessa memória, pois são construídas através do tempo, fundamentadas na análise de um histórico de experiências boas e ruins da rede, e não tendem a mudar rapidamente e nem constantemente.

Aqui é válida uma correlação do mundo real com a política de memórias empregada no Of-IDPS. Tomando como exemplo uma pessoa que vai trabalhar pela primeira vez com fundição de aço em uma metalúrgica, ou seja, não conhece o ambiente de trabalho. É bem provável que em um primeiro momento essa pessoa coloque a mão em algum lugar extremamente quente e queime a mão. Nesse instante irá entrar em ação a memória sensorial, pois assim que os sensores acusarem o excesso de calor e dor (alerta), serão criadas regras de segurança que determinam a ação da retirada imediata da mão do lugar quente. Note que a memória sensorial não fará com que a pessoa mantenha a ação de mover sua mão rapidamente durante o resto da vida da pessoa. Ao contrário, essa ação é aplicada e removida bem rapidamente, só para resolver o problema imediato da mão naquele lugar quente. Bem, mesmo com o perigo de queimar a mão a pessoa continuará no ambiente hostil, pois é seu

trabalho. Então, entra em ação a memória curta, que rapidamente deve avaliar os acontecimentos (alertas) recentes e estipular padrões (regras) para que a pessoa não se machuque mais. Desta forma, a memória curta deve, utilizar os alertas dos sensores para perceber qual é o calor que ela pode aguentar sem sofrer danos, ou seja, através dos alertas gerados pelos sensores a pessoa vai analisando a situação do ambiente até estabelecer regras para manter a sua integridade física. Com o passar do tempo, a pessoa trabalhando na metalúrgica irá fixar/aprender o que representa perigo real, bem como, o que não é tão ameaçador na fundição de aço. Ou seja, a memória longa terá criado o conhecimento necessário (regras de segurança) para evitar situações de risco a integridade física da pessoa. Note também, que os padrões estabelecidos na memória longa não têm propensão à mudar rapidamente, pois foi/é construído ao longo do tempo, através da análise dos alertas e situações vividas e analisadas pelas memórias sensoriais e curta.

Por fim, a memória curta do Of-IDPS é executada de forma ininterrupta e continua por uma *thread* (Tanenbaum, 2007), criando e mantendo regras de segurança que são armazenadas na base de dados Regras Memória Curta, que é acessada pelo módulo Execução para que as devidas ações sejam tomadas frente aos riscos eminentes à rede.

4.2.3.3.3 Memória longa

A memória longa é responsável por aprender o que é bom ou ruim para a rede e isso é feito analisando-se o histórico de uso da rede, bem como, os transtornos já enfrentados.

Diferente das memórias sensorial e curta, do Of-IDPS, que só analisam problemas/alertas de segurança. A memória longa proposta aqui, também examina os dados de uso da rede obtidos através dos sensores OpenFlow, para então determinar o que é comum na rede.

Então, na prática o Of-IDPS possui dois submódulos de memória longa (ver Figura 4.6):

- **Ruim**: Tem por objetivo identificar as principais ameaças que afligem a rede. Como isso é feito através da análise de lembranças ruins (alertas de segurança), tal memória é chamada de memória longa ruim;
- **Boa**: Ajuda a identificar quais são os principais serviços e *hosts* utilizados na rede. Para isso são analisados apenas os fluxos de rede considerados honestos/legítimos e devido isso a presente memória é chamada de memória longa boa. São considerados fluxos de rede honestos aqueles que não estão diretamente relacionados com alertas de segurança.

Tanto a memória longa ruim quanto a boa empregam o algoritmo FP-Growth e os métodos descritos na Subseção 4.2.3.2 para determinar os padrões de ataques e de fluxos comuns da rede. As memórias longas também são executas regularmente por uma *thread* (Tanenbaum, 2007). As Figuras 4.14 e 4.15 apresentam respectivamente, o fluxo de processamento das memórias longa ruim e boa.

Memória longa ruim

Conforme pode ser observado na Figura 4.14, a memória longa ruim, tal como as memórias sensorial e curta, tem acesso aos alertas de segurança mantidos pelo módulo Monitor, a única diferença é que esta trata de alertas mais antigos. Todas as regras de segurança criadas pela memória são armazenadas na base de dados Regras Memória Longa Ruim para posteriormente ficarem a disposição do módulo Execução, que converterá as regras de segurança em ações.

Ainda analisando a Figura 4.14, observa-se que o processamento da memória longa ruim inicia com uma função que apaga todas as regras de segurança previamente existentes na base Regras Memória Longa Ruim. Em seguida, uma função obtêm o histórico de alertas de segurança. Caso não sejam retornados alertas, a memória longa ruim fica sem regras de segurança e reinicia o processo. Mas, caso sejam retornados alertas, esses são submetidos ao algoritmo FP-Growth e as regras indesejadas são filtradas pelos métodos explicados na Subseção 4.2.3.2. Por fim, as regras de

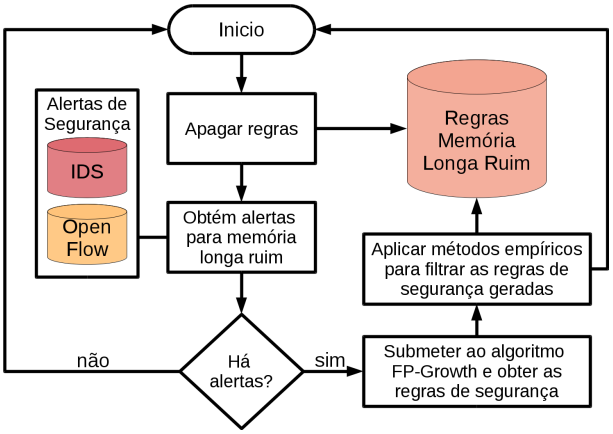


Figura 4.14: *Geração de regras de segurança na memória longa de lembranças ruins.*

segurança geradas são gravadas na base de dados Regras Memória Longa Ruim e todo o processo recomeça.

Memória longa boa

A memória longa boa, funciona de maneira similar a memória longa ruim (ver Figura 4.15), exceto pelo fato de que na memória boa, ao invés de alertas de segurança, são obtidos dados a respeito do uso rede, extraídos dos sensores OpenFlow e que estão armazenados na base de dados Fluxos Normais do submódulo Dados de Uso da Rede do módulo Monitor (Figura 4.6). Os dados extraídos da base Fluxos Normais possuem basicamente os mesmos campos dos alertas de segurança, explicados na Subseção 4.2.2.0.1.

As regras de segurança resultantes da memória longa boa representam o cotidiano da rede, ou seja, fluxos comuns. Então, outra diferença fundamental, é que as regras geradas aqui não servem para restringir largura de banda ou bloquear pacotes de rede. Ao contrário, são para liberar a passagem dos fluxos que combinem com as regras, pois essas são a representação de serviços e *hosts* idôneos na rede. As regras de segurança resultantes do processamento dos métodos da memória longa boa são armazenadas em uma base de dados chamada Regras Memória Longa Boa e ficam acessíveis ao módulo Execução.

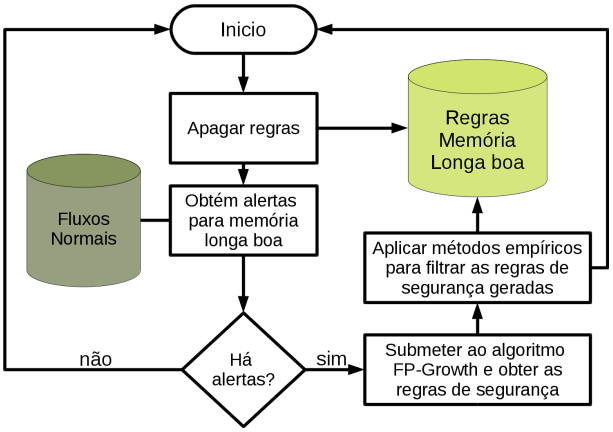


Figura 4.15: *Geração de regras de segurança na memória longa de lembranças boas.*

Métodos de busca de alertas na memória longa

Como o problema de processar conjuntos de itens frequentes é um problema exponencial e os alertas/fluxos de redes processados pela memória longa tendem a ter grandes volumes de dados. A função de obter alertas/fluxos de rede para serem processados pelas memórias longa ruim e boa, implementa várias formas diferentes de buscas. Assim, caso necessário, é possível optar por tipos diferentes de buscas, tal como:

- Buscar todos alertas/fluxos existentes - retornará todos os alertas/fluxos existentes o que deve proporcionar uma ótima análise do comportamento da rede, mas em contra partida exigirá muito processamento caso haja um grande montante de dados;

- Buscar os últimos N alertas/fluxos - trabalhará apenas com as últimas informações a respeito da rede. Desta forma, o sistema acabará "esquecendo" o que ficou no passado distante, porém dá para adequar perfeitamente o total de alertas/fluxos retornado com a capacidade de processamento do sistema;

- Buscar os alertas/fluxos dos últimos N minutos - esta busca é parecida com a anterior, porém um pouco mais dinâmica, já que a quantidade de alertas pode aumentar ou diminuir no tempo. Esse comportamento dinâmico pode levar a duas situações indesejadas: (1) se no período pré-estabelecido chegar uma quantidade gigantesca de alertas/fluxos, o sistema pode sofrer sobrecargas; ou (2) se nesse período não houver nenhum alerta/fluxo, o sistema não terá como analisar o comportamento da rede. Entretanto, este método pode ser visto como uma vantagem, principalmente em redes que mudam constantemente suas características (*hosts*, serviços, etc), pois assim a memória longa do Of-IDPS "esquecerá" mais rapidamente o velho e tenderá para o novo;

- Buscar os aletas usando métodos de amostragem estatística - retornará uma amostra da população para descobrir o padrão de ataques e fluxos comuns da rede. A vantagem é que a amostragem normalmente representa uma porcentagem pequena da população, o que deve poupar processamento. A desvantagem é que a amostra pode não representar adequadamente toda a população.

Então, é possível configurar o Of-IDPS com o método de busca que melhor se adapta a realidade da rede. É válido lembrar que o sistema não deixará de ser protegido caso o método de busca da memória longa seja empregado erradamente, pois o Of-IDPS também emprega as memórias sensoriais e curta para proteger a rede. Além do mais, a memória longa não necessita ser processada instantaneamente, já que como foi discutido em toda a Seção, esta tende a mudar lentamente. Assim, os diferentes métodos de busca propostos são apenas uma opção de refinamento do sistema.

Voltado a memória longa boa, um dos principais objetivos da sua idealização neste trabalho é evitar que regras de segurança, criadas por falsos positivos ou mesmo por ataques reais, interfiram em serviços vitais da rede. Então, as respostas autonômicas contra incidentes de segurança do Of-IDPS tentam não interromper os serviços utilizados frequentemente na rede. Pois, o bloqueio de serviços legítimos, quando se utiliza respostas automáticas contra incidentes de segurança é um problema conhecido na área da segurança da informação e por isto nosso trabalho resolveu atacá-lo propondo a identificação de padrões de fluxos importantes que não devem ser perturbados.

Na prática as regras da memória longa boa, serão contrapostas com as regras "ruins" geradas pelas memórias: curta, sensorial e longa ruim. A Seção seguinte dará mais detalhes a respeito de como utilizar a memória boa para evitar principalmente o corte abrupto de serviços conhecidos da rede.

Finalmente, com o esquema de memórias de segurança somado as técnicas de aprendizado de máquina, a rede terá capacidade de assimilar o que é bom ou ruim. Fundamentado nisso criará regras para se auto-proteger contra ameaças à segurança. A seção seguinte detalha como as regras de segurança criadas são aplicadas na rede, para tornar o sistema autonômico completo.

4.2.4 Execução no Of-IDPS

O módulo Execução é responsável por traduzir, as regras que compõem a política de segurança, em comandos a serem executados pelos atuadores, visando aplicar contramedidas na rede para mitigar problemas de segurança.

No Of-IDPS o módulo Execução tem três tarefas básicas:

1. Converter regras de segurança em contramedidas para mitigar problemas (submódulo "Aplicar contramedida" da Figura 4.6);
2. Impor a política de segurança aos fluxos de rede já existentes. Isto é, aplicar as regras de segurança do administrador e as regras de segurança autonômicas (memórias sensorial, curta, longa ruim e boa) aos fluxos já instalados na rede (passo 5.1 da Figura 4.6);
3. Comutar os novos pedidos de inclusão de fluxo de rede que são enviados ao controlador OpenFlow segundo a política de segurança (passo 5.2 da Figura 4.6).

As Subseções 4.2.4.1, 4.2.4.2 e 4.2.4.3, a seguir explicam com mais detalhes essas três tarefas básicas do módulo Execução do Of-IDPS.

4.2.4.1 Contramedidas no Of-IDPS

Um ato muito importante no módulo Execução é a vinculação do campo prioridade da regra de segurança[7] com uma contramedida que efetivamente altere o comportamento da rede para resolver desequilíbrios.

Neste trabalho defini-se como contramedida uma ação prática que tem por objetivo alterar o estado atual da rede ou de partes dessa (fluxos, *hosts*, serviços, etc). Sendo que no Of-IDPS, há três categorias de contramedidas previstas:

- **Redirecionar**: Redireciona fluxos de rede. Essa contramedida pode ser útil no caso de incertezas sobre a idoneidade de determinados fluxos de rede. Então, fluxos duvidosos podem ser desviados para ambientes como *honeypots* ou *honeynets*, com o objetivo de averiguar se são maliciosos ou não. Em outras palavras, transações de rede duvidosas podem ser colocadas em quarentena para averiguação;
- **Alterar Largura de Banda**: Permite modificar a largura de banda de elementos como: fluxos, serviços, *hosts*, para equalizar o uso da rede. Aqui usa-se largura de banda como sinônimo de taxa de transferência de dados (Tanenbaum e Wetherall, 2013). Essa contramedida obriga os fluxos suspeitos a usarem os recursos da rede de forma limitada/dosada. A principio, o Of-IDPS utiliza três tipos de largura de banda pré-definidas:
  - Restringir suavemente: Limita a taxa de transferência de dados brandamente. Isso é configurável, mas deve ser algo como 50% da largura de banda da rede;
  - Restringir severamente: Reduz drasticamente a taxa de transferência de dados. Também é configurável, mas o ideal é que seja utilizada uma largura de banda que torne o uso da rede quase impraticável, tal como alguns poucos Kbps (*Kilobits* por segundo). O objetivo é desestimular o ataque tornando o acesso a rede lento ao ponto de frustrar o atacante. Esse tipo de contramedida é interessante, pois pode mitigar o ataque sem dar pistas de que o sistema de segurança existe. Além de que pode ser considerada uma contramedida menos rígida do que bloquear;
  - Não restringir: Não aplica restrições de largura de banda quanto ao uso da rede, ou seja, dá a oportunidade de utilizar 100% da rede. Essa contramedida pode ser utilizada para restabelecer a largura de banda total para fluxos que tiveram sua taxa de transferência de dados reduzida no passado.

[7] Descrito na Subseção 4.2.2.1

- **Excluir/Bloquear**: Remove fluxos de rede. Essa contramedida irá deletar os fluxos maliciosos dos *switches* OpenFlow. Caso os fluxos removidos tentem posteriormente restabelecer conexão, esses podem ser bloqueados pela comutação do Of-IDPS, que será explicada posteriormente nesta seção.

Desta forma, a conversão regras de segurança em contramedidas, segue por padrão os seguintes preceitos:

- Regras de segurança com prioridade alta empregam a exclusão seguido do bloqueio dos fluxos de rede;
- Regras com prioridade média regulam os respectivos fluxos de rede com uma restrição severa da largura de banda;
- Regras com prioridade baixa aplicam restrições suaves a taxa de transferência de dados dos fluxos que combinam com essas regras;
- Quando não há prioridade vinculada a regra (0 – zero), como é o caso das regras formadas pela memória longa boa, os fluxos que combinarem com essas regras são configurados para fluir normalmente pela rede, podendo usar 100% da largura de banda disponível.

Atualmente nenhum nível de alerta de segurança está vinculado a contramedida redirecionamento, mas essa foi prevista e futuramente poderá ser aplicada em casos de dúvida a respeito de determinados comportamentos na rede. No entanto, as outras contramedidas são empregadas constantemente tanto para alterar o comportamento de fluxos já instalados na rede, quanto na comutação de novos fluxos no Of-IDPS, que são respectivamente os passos 5.1 e 5.2 da Figura 4.6 e serão explicados a seguir.

4.2.4.2 Imposição das regras de segurança em fluxos já instalados/ativos na rede

Tão logo as regras de segurança sejam geradas pela arquitetura autonômica, essas devem ser aplicadas aos fluxos já existes/ativos para remediar anomalias que afetem o desempenho da rede.

A tarefa de impor as regras de segurança em fluxos já ativos na rede é ilustrada brevemente pelo passo 5.1 da Figura 4.6. Contudo, é melhor detalhada na Figura 4.16, que ilustra a referida tarefa iniciando com a obtenção das regras de segurança que estão disponíveis na base de dados Regras de Segurança do Administrador e nas bases das memórias sensorial, curta, longa boa e longa ruim (essas são as bases de dados que formam a política de segurança do módulo Base de Conhecimento da arquitetura proposta – ver Figura 4.7).

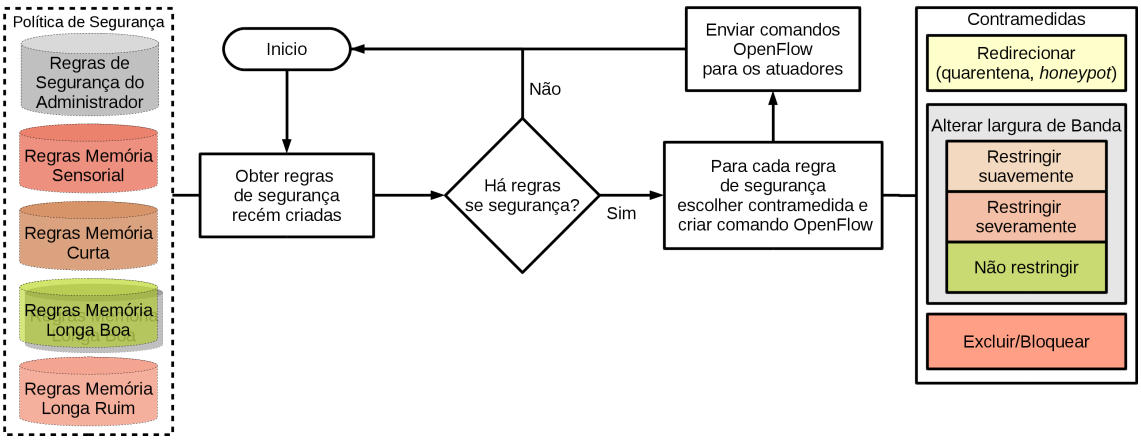


Figura 4.16: *Conversão de regras em contramedidas de segurança para os fluxos já instalados na rede.*

Segundo o fluxo de processamento apresentado na Figura 4.16, nada será feito caso não haja regras novas disponíveis nas memórias de segurança do Of-IDPS. No entanto, caso existam novas regras de segurança, essas serão convertidas em comandos OpenFlow que serão enviados e executados pelos atuadores.

Por exemplo, caso sejam retornas as regras de segurança da Tabela 4.3, essas serão convertidas para suas respectivas contramedidas e terão as seguintes interpretações:

- Regra 1: Restringir a largura de banda suavemente para os fluxos ICMP do tipo *echo request* (*type* 8 e *code* 0 (Tanenbaum e Wetherall, 2013)) destinados ao *host* com IP 10.0.0.1;
- Regra 2: Restringir a largura de banda severamente para o fluxo de rede com IP de origem 200.0.0.2, porta de origem 55.000, IP de destino 10.0.0.2, porta de destino 80 e protocolo TCP;
- Regra 3: Excluir fluxos TCP com destino a porta 23 do *host* 10.0.0.3.

Tabela 4.3: *Exemplo de regras de segurança a serem convertidas em contramedidas.*

N°	Endereço IP		Protocolo	Portas de rede		Alerta de segurança		Suporte
	Origem	Destino		Porta Origem	Porta destino	Identificação	prioridade	
1	*	10.0.0.1	ICMP	8	0	*	baixa	1000
2	200.0.0.2	10.0.0.2	TCP	55000	80	DoS	média	900
3	*	10.0.0.3	TCP	*	23	Brute Force	alta	1200

Dado o exemplo anterior e as explicações da subseção anterior (4.2.4.1), fluxos que combinem com regras de segurança com prioridade de segurança baixa terão seu fluxo de transferência de dados reduzido suavemente. Assim, quando o comando derivado da regra 1, for submetido a rede, todos os fluxos que combinarem com esse terão sua taxa de transferência de dados reduzida brandamente. Da mesma forma, o comando produzido a partir da regra de segurança 2, fará com que os fluxos associados aos campos da regra tenham sua largura de banda limitada drasticamente, pois é a contramedida aplicada caso o nível de prioridade seja médio. Já a regra 3 da Tabela 4.1, irá fazer com que os fluxos que combinem com o comando produzido sejam removidos dos *switches* OpenFlow e posteriormente bloqueados pelo controlador, caso tentem se restabelecer na rede (a próxima subseção explica com mais detalhes como o bloqueio é realizado).

No módulo Execução, todos os comandos OpenFlow criados a partir das regras de segurança serão executados pelos atuadores. Então, os ditos comandos são mensagens OpenFlow, que serão enviadas do controlador OpenFlow (onde está instalado o módulo Execução) para os *switches* OpenFlow, que fazem o papel dos atuadores.

Emfim, os *switches* OpenFlow recebem as mensagens OpenFlow e as executam prontamente alterando os fluxos já instalados para harmonizar a rede.

4.2.4.3 Comutação no Of-IDPS

O módulo Execução também tem como tarefa básica comutar os novos fluxos obedecendo as regras de segurança redigidas pelo administrador e criadas autonomicamente pelas memórias do Of-IDPS.

A comutação de pacotes no Of-IDPS é realizada tal como no Algoritmo 3, que a principio opera de forma similar a comutação aplicada em *switches* comuns (Tanenbaum e Wetherall, 2013), só que utilizando a tecnologia OpenFlow.

A principal diferença na comutação de pacotes no Of-IDPS, é que antes do controlador permitir a instalação de um fluxo para o encaminhamento do pacote analisado e seus subsequentes, esse deve verificar se os pacotes do fluxo estão alinhados com a política de segurança do módulo Base de Conhecimento. Em outras palavras, o controlador deve verificar se alguma regra de segurança redigida manualmente pelo administrador da rede ou autonomicamente pelas memórias sensorial,

curta, longa boa ou longa ruim, combinam com os campos do dito pacote que está entrando no controlador OpenFlow (passo 5.2 da Figura 4.6), a partir dai há duas possibilidades:

- Caso combinem, a contramedida equivalente com a prioridade de segurança da regra deve ser aplicada ao pacote que está entrando e ao referente fluxo. Assim, o fluxo relacionado com o pacote sendo analisado pelo controlador, pode ser:

  1. Instalado normalmente caso combine com alguma regra que libere/permita sua passagem normalmente na rede. Essa regra pode ser criada autonomicamente pela memória longa boa ou ser redigida manualmente pelo administrador da rede;
  2. Instalado com restrições de largura de banda. Essa regra pode ser criada autonomicamente pelas memórias sensorial, curta e longa ruim ou ser redigida manualmente pelo administrador da rede;
  3. Bloqueado completamente. Essa regra pode ser criada autonomicamente pelas memórias sensorial, curta e longa ruim ou ser redigida manualmente pelo administrador da rede.

- Caso o pacote não combine com nenhuma regra de segurança, esse e o respectivo fluxo serão encaminhados normalmente pelo controlador e consequentemente pelos *switches* OpenFlow.

O Algoritmo 3 apresenta com mais detalhes a ideia da comutação empregada no Of-IDPS. Sendo possível observar que o encaminhamento de pacotes no Of-IDPS é feito da seguinte forma:

1. A comutação inicia com a chegada de um pacote no controlador OpenFlow que é recepcionado pelo método *recebePacotesEntrandoNoSwitch()* e armazenado no objeto *pacoteEntrando*. Conforme a Seção 2.3.1, pacotes são enviados para o controlador quando não combinam com nenhum fluxo previamente instalado nos *switches* OpenFlow;

2. O controlador OpenFlow registra a tupla: endereço de origem do pacote; e a porta do *switch* por qual o pacote entrou. Esse é um procedimento comum na comutação de pacotes e ajuda posteriormente na localização da máquina que originou o pacote. Tal procedimento é realizado na linha 2 do algoritmo;

3. Na linha 3, utilizando o endereço de destino do pacote em questão, o controlador verifica se é conhecida a porta do *switch* em que se encontra o *host* de destino, para onde o pacote deve ser encaminhado. A informação resultante é armazenada na variável *portaSaida*;

4. O próximo passo (linha 4) é verificar o conteúdo de *portaSaida*. Dependendo do valor armazenado há duas possibilidades:

   (a) Se a variável for nula (ϕ), significa que o controlador não sabe em qual porta está o *host* de destino que o pacote tenta alcançar. Então, o processamento do pacote termina com esse sendo enviado para todas as portas do *switch* (método *enviarParaTodasPortasDoSwitch()* - linhas 15 e 16). Esse também é um comportamento comum na comutação de pacotes tradicional;

   (b) Senão, a variável *portaSaida* contém o valor que representa a porta de destino do *switch* para onde o pacote deve ser encaminhado. Com essa informação o controlador, a princípio, já tem condições de enviar uma mensagem OpenFlow pedindo que o *switch* instale um fluxo de rede para que o pacote sendo analisado e seus subsequentes alcancem o seu destino - esse seria o procedimento adotado para que a rede OpenFlow funcione de forma similar a tradicional comutação de pacotes Ethernet. Todavia no Of-IDPS, antes do controlador instalar esse fluxo no *switch*, o pacote em questão ainda será comparado com as regras de segurança do Of-IDPS:

- Desta forma, na linha 5, o método *verificaSeCombinaComRegrasDaPolíticaDeSegurança()* é utilizado para submeter o pacote frente as regras de segurança. O resultado desse processamento será a prioridade de segurança da regra que combinar com o pacote sendo analisado e tal prioridade será armazenada na variável *ação*;
- Em seguida, o sistema analisa a prioridade de segurança armazenada no campo *ação* e fundamentado nela emprega uma das 4 possibilidades de encaminhamento (linhas 6-14) :
  - O fluxo será instalado normalmente e sem restrições (linhas 7 e 8), caso o valor da variável *ação* seja igual ao valor da prioridade de segurança NÃO_HÁ. Este valor é retornado caso o pacote não combine com as regras de segurança de nenhuma memória, ou caso combine perfeitamente com uma regra da memória longa boa ou com uma regra redigida pelo administrador liberando a passagem do pacote pela rede. Em outras palavas, o controlador instalará nos *switches* OpenFlow os fluxos que não estejam relacionados com problemas de segurança ou sejam considerados comuns pela memória boa;
  - Caso o valor da variável *ação* seja igual a *BAIXA* o fluxo será instalado no *switch*, porém com uma branda restrição de largura de banda (linhas 9 e 10). O valor só será *BAIXA* caso os pacotes analisados combinem com alguma regra que possua a prioridade de segurança baixa, desta forma a contramedida equivalente será aplicada;
  - Caso a *ação* seja *MÉDIA*, o referido fluxo do pacote ainda será instalado, contudo com uma drástica limitação em sua largura de banda (linhas 11 e 12), pois coincidiu com uma regra com prioridade de segurança média;
  - Se a variável *ação* for *ALTA*, o pacote sendo analisado será simplesmente descartado e nenhum fluxo será instalado para esse nos *switches*. Isto é, o pacote e seus subsequentes são bloqueados. Essa atitude extrema só é tomada, pelo Of-IDPS, quando os pacotes em análise combinam com alguma regra que representa alto risco contra a segurança da rede.

Algoritmo 3: Comutação utilizada no Of-IDPS

```
   Entrada: Primeiro pacote de rede de cada transmissão
   Saída: Libera/altera largura de banda/bloqueia o fluxo OpenFlow
1  pacoteEntrando ← recebePacotesEntrandoNoSwitch()
2  tabelaMAC ← pacoteEntrando.registrarPortaDeEntradaDoPacote(pacoteEntrando)
3  portaSaida ← tabelaMAC.buscarPortaSaida(pacoteEntrando)
4  se portaSaida ≠ ϕ então
5      acao ← verificaSeCombinaComRegrasDaPolíticaDeSegurança(pacoteEntrando)
6      selecione acao faça
7          caso prioridade.NÃO_HÁ
8              enviaPacoteEInstalaFluxoNormalmente(pacoteEntrando)
9          caso prioridade.BAIXA
10             enviaPacoteEInstalaFluxoComRestriçãoDeLarguraDeBandaSuave(pacoteEntrando)
11         caso prioridade.MÉDIA
12             enviaPacoteEInstalaFluxoComRestriçãoDeLarguraDeBandaSevera(pacoteEntrando)
13         caso prioridade.ALTA
14             bloqueiaPacote(pacoteEntrando)
15 senão
16     enviarParaTodasPortasDoSwitch(pacoteEntrando)
```

4.2.4.3.1 Análise das regras de segurança durante a comutação

Todo o processamento realizado pelo Algoritmo 3 e principalmente a submissão dos pacotes as regras da política de segurança, executado pelo método *verificaSeCombinaComRegrasDaPolíticaDeSegurança()*, pode ser melhor visualizado através da Figura 4.17.

Então, a Figura 4.17, ilustra que o processo inicia com o pacote chegando da rede ao controlador OpenFlow. Na sequencia o controlador, através do módulo Execução, submete o pacote as regras de segurança redigidas manualmente pelo administrador da rede. Ou seja, compara o pacote com as regras de segurança que estão armazenadas na base de dados Regras de Segurança do Administrador.

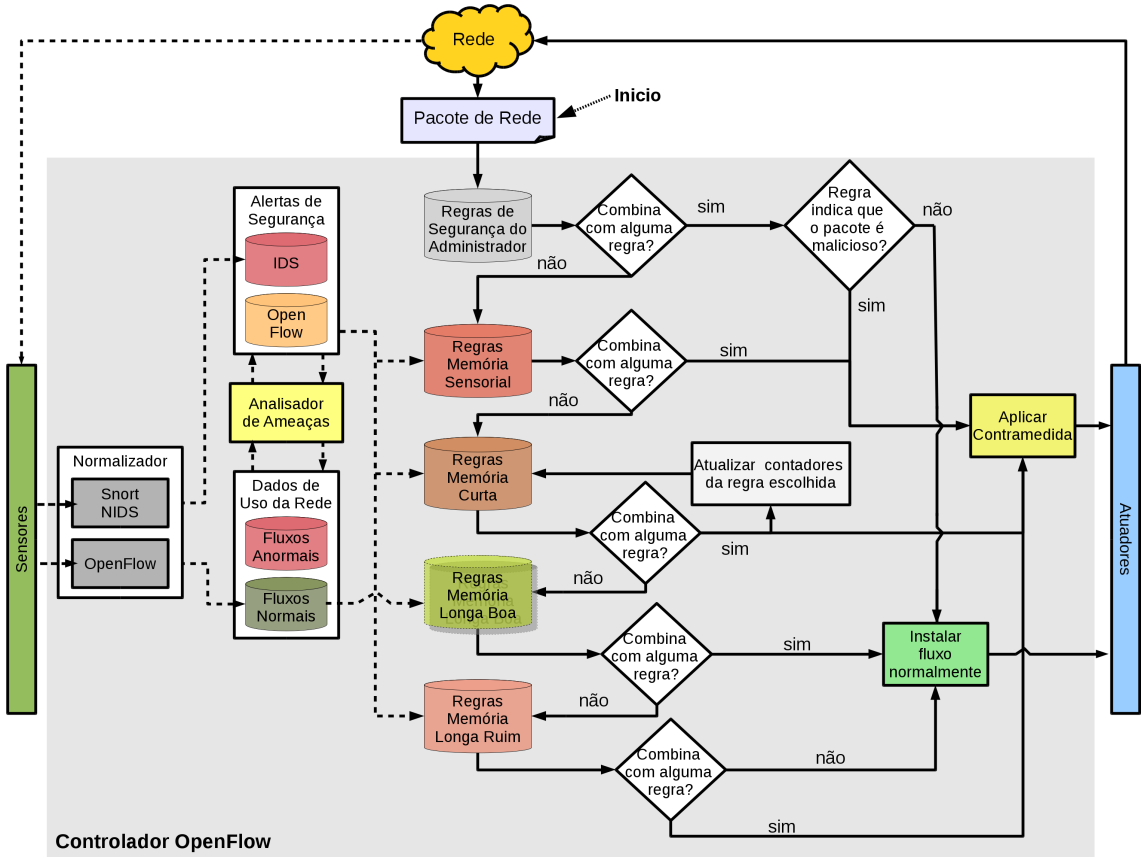


Figura 4.17: *Exemplo de fluxo que um pacote segue quando é submetido as regras da política de segurança do Of-IDPS.*

A comparação do pacote com cada regra se dá através dos campos: IP origem, porta de origem, IP destino, porta de destino e protocolo. Se ao final da comparação com as regras de uma dada base de dados for constatado que o pacote combina com mais de uma regra, a seguinte lógica é utilizada para o desempate: (1) ganha a regra que combine mais perfeitamente com a maior quantidade de campos do pacotes, sem contar os campos com * (que combinam com todos). (2) caso o impasse continue aplica-se a regra que representa mais risco a rede (maior prioridade de segurança). (3) persistindo o empate é aplicada a primeira regra dentre as restantes (tal como descrito na Subseção 4.2.3.2).

Voltando a Figura 4.6, caso o pacote em análise combine com alguma regra da base de dados Regras de Segurança do Administrador, tal regra será imediatamente aplicada ao pacote, desta forma o fluxo será submetido a contramedida da regra que combinou com o pacote e o processamento termina. Ou seja, esse pacote não será submetidos as outras bases de dados de regras de segurança (no caso as bases das memórias sensorial, curta, longa boa e ruim). Esse comportamento é padrão e válido para qualquer outra base de regras de segurança. Então a primeira base que combinar com o pacote sendo processado vence.

Supondo, que no exemplo da Figura 4.6, nenhuma regra redigida pelo administrador da rede case com o pacote em análise, esse vai ser submetido as regras de segurança da memória sensorial. Então, caso algum pacote case com uma regra de segurança dessa memória a referida contramedida será aplicada, mas caso nenhuma regra combine o pacote é submetido as regras da memória curta.

No caso do pacote coincidir com alguma regra da memória curta, essa regra será automaticamente atualizada para ganhar mais tempo de vida no sistema - isto mostra que a regra está sendo

útil, tal como explicado na Subseção 4.2.3.3.2.

Entretanto, caso o pacote não combine com as regras da memória curta, esse será processado pelas regras da memória longa boa. Conforme já foi explicado, diferentemente das outras memórias, se o pacote combinar com alguma regra da memória boa esse não será reprimido, mas sim prontamente instalado na rede.

Todavia, caso o pacote ainda não combine com as regras da memória longa boa, esse será submetido a memória longa ruim. Se combinar com alguma regra dessa memória, então será aplicada alguma contramedida. Mas, caso não combine com nenhuma regra de segurança, um fluxo OpenFlow será instalado para que o pacote e seus subsequentes utilizem a rede livremente. Desta forma, caso um pacote em análise, no controlador OpenFlow, não case com nenhuma regra das bases de dados que formam a política de segurança, esse pacote por padrão é instalado sem restrições na rede.

Ordem de processamento da memória longa boa

Outro ponto a ser observado na análise de regras das memórias do Of-IDPS, é que a ordem de análise da memória longa boa em relação as outras memórias é configurável. Isso significa que ela pode ser posta a frente ou entre qualquer uma das memórias. O sombreado na base de dados Regras Memória Longa Boa da Figura 4.17 representa essa possibilidade. Desta forma, as seguintes configurações no arranjo entre as memórias são plausíveis no Of-IDPS:

1. **Longa boa**, sensorial, curta e longa ruim;
2. Sensorial, **longa boa**, curta e longa ruim;
3. Sensorial, curta, **longa boa** e longa ruim;
4. Sensorial, curta, longa ruim e **longa boa**;

A possibilidade de mudança na ordem de análise da memória longa boa foi realizada pensando-se em deixar o Of-IDPS flexível o bastante para que este se adéque aos mais diferentes tipos de redes e principalmente as necessidades dos administradores. É claro que cada configuração tem vantagens e desvantagens que devem ser levadas em consideração pelo administrador da rede, por exemplo levando em consideração as ordens de memória apresentadas anteriormente:

- Na configuração 1, os fluxos tidos como bons nunca serão afetados pelas contramedidas das outras memórias, o que pode ser usado para evitar problemas de falsos positivos. Todavia, se ocorrer um ataque em um desses fluxos "bons", esse ataque não será imediatamente mitigado, pois estará amparado pelas regras da memória longa boa;
- Na configuração 2, os ataques muito recentes podem ser combatidos, mesmo que ocorram em fluxos considerados "bons", mas só de forma pontual pela memória sensorial. Ou seja, a mitigação de ataques em fluxos bons é realizada com essa configuração, mas de forma acanhada, pois é utilizada apenas a reação de curtíssimo prazo da memória sensorial e principalmente não são empregadas as técnicas de aprendizado da memória curta para combater os males que podem aparecer nos fluxos comuns protegidos pela memória longa boa.
- A configuração 3, simboliza o que normalmente se espera da memória humana. Isto é, o sistema a principio terá ou formará conhecimento prévio a respeito do que é bom e ruim (memórias longa boa e ruim). De qualquer forma, se forem observadas anomalias/perigo no que o sistema julgava comum e sem riscos (regras memória longa boa), o Of-IDPS não irá hesitar em aplicar as contramedidas criadas em curtíssimo e curto prazo (memórias sensorial e curta) para manter a rede segura. Além do que se o problema persistir durante muito tempo, o Of-IDPS irá inclusive aprender e mudará a sua conclusão a respeito do que antes era considerado bom ou ruim.

- A configuração 4, na prática não existe. Mas, exite a possibilidade de desligar a memória longa boa no Of-IDPS, o que seria equivalente a essa configuração. É claro que desativando a memória longa boa, o sistema perde a capacidade de identificar o que é normal/comum na rede e terá mais dificuldade em proteger-se de falsos positivos.

É importante notar que a troca da ordem de análise da memória longa boa, só afeta a forma de reação frente aos fluxos considerados ordinários e não compromete a capacidade do Of-IDPS em combater fluxos de redes considerados anormais.

Quanto a ordem de análise das regras de segurança do administrador, sempre que um pacote chegar no sistema esse pacote será primeiramente comparado com essas regras (ver Figura 4.17). A ordem de análise dessa base é imutável/fixa. Isso garante que as vontades do administrador sejam respeitadas pelo sistema autonômico. Ou seja, um pacote só será comparado com as regras de segurança das memórias do Of-IDPS, se não casar com nenhuma regra da base de dados Regras de Segurança do Administrador, que é obrigatóriamente o primeiro conjunto de regras que um pacote entrando no controlador OpenFlow é submetido.

Finalmente, é o módulo Execução, através dos métodos explicados anteriormente, que permite converter tudo o que foi planejado pela arquitetura autonômica do Of-IDPS, em ações concretas a serem executadas pelo controlador ou pelos *switches* OpenFlow, para mitigar problemas de segurança que surgem na rede.

4.2.5 Extração de alertas de segurança postados em redes sociais

Devido ao surgimento constante de ameaças contra a segurança de ambientes computacionais, é fundamental que administradores de redes se mantenham bem informados a respeito dos perigos aos quais as redes estão expostas. Ter acesso rápido a informações que relatem ameaças contra redes de computadores mundo a fora permite, por exemplo, que administradores adaptem/atualizem os mecanismos de segurança para proporcionar um combate mais eficiente contra as ameaças que surgem diariamente. Então, este trabalho propõem o uso de informações externas à rede de computadores, para auxiliar na identificação, evidenciação ou elucidação de problemas de segurança que prejudicam ou poderão prejudicar a rede local.

Atualmente as redes sociais são uma grande fonte de informação e ajudam a disseminar qualquer tipo de notícia rapidamente (Russell, 2011), inclusive as relacionadas com segurança de computadores (Campiolo *et al.*, 2013; Santos *et al.*, 2012, 2013). Desta forma, a arquitetura proposta através do módulo Base de Conhecimento propõem a extração de alertas de segurança a partir de mensagens postadas no *microblog* Twitter, para auxiliar os administradores de redes, no processo de identificação de problemas de segurança. O Twitter foi escolhido devido principalmente a sua política de privacidade, que basicamente permite que qualquer pessoa visualize/acesse a postagem de qualquer usuário dessa rede social, ou seja, as postagens são públicas. Além do mais, o Twitter é uma rede social muito utilizada como fonte de notícias, principalmente na disseminação de informações a respeito de acontecimentos atuais/recentes (Cha *et al.*, 2012; Lerman e Ghosh, 2010), o que pode ajudar na identificação de novas ameaças contra redes de computadores.

A tarefa de extrair alertas de segurança a partir de mensagens postadas no Twitter é realizada pelo submódulo Vulnerabilidades Externas do módulo Base de Conhecimento (ver Figura 4.3). A metodologia utilizada nessa tarefa é melhor detalhada na Figura 4.19 e é composta por quatro passos:

1. Obter do Twitter, os *tweets* postados contendo textos relacionados com ameaças contra a segurança de computadores;

2. Filtrar os *tweets* coletados, objetivando remover mensagens/*tweets* que não estão diretamente relacionadas com segurança de computadores. Tal como *spam*, mensagens mal formadas ou fora de contexto (por exemplo, que relatam guerras entre povos, culturas e/ou religiões);

3. Agrupar as mensagens de acordo com a similaridade de seus textos (Manning *et al.*, 2008), ou seja, agrupar *tweets* que abordam assuntos parecidos;

4. Gerar lista com os grupos/mensagens mais relevantes através da análise de características dos *tweets* que formam os grupos.

Ao final do quarto passo (Figura 4.19) as mensagens mais relevantes, evidenciadas pela metodologia, são salvas na base de dados Ameaças Citadas em Redes Sociais do módulo Base de Conhecimento (Figura 4.3) como alertas de segurança. E podem ser empregados como fonte extra de informações para evidenciar problemas de segurança em redes de computadores. As subseções a seguir irão detalhar melhor os passos utilizados nessa metodologia.

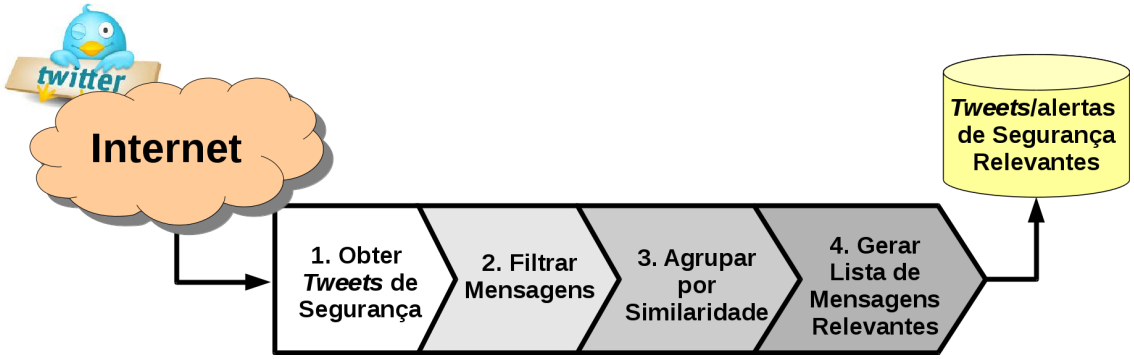


Figura 4.18: *Método para extrair alertas de segurança do Twitter (Santos et al., 2013).*

4.2.5.1 Obter *tweets* relacionados com segurança de computadores

A obtenção/coleta de *tweets* relacionados com segurança de computadores consiste basicamente em submeter termos de buscas ao Twitter e salvar os *tweets* retornados em uma base de dados para posteriormente processá-los.

Os *tweets* são obtidos do Twitter através de buscas bem direcionadas à área de segurança de computadores. Nessas buscas são utilizados termos e expressões como:

1. *security AND (virus OR worm OR attack OR intrusion OR invasion OR ddos OR hacker OR cracker)*;
2. *security AND (vulnerability OR exploit OR patch OR malware)*;
3. *zero-day AND (vulnerability OR exploit OR attack)*;
4. *security AND (trojan OR botnet OR hijack OR backdoor)*;
5. *security alert AND (windows OR linux OR apple OR android OR java OR adobe)*;
6. *spyware*;
7. *security AND (overflow OR bug)*;
8. *dns poisoning OR fake browse window OR malicious browser extensions OR malicious proxy*.

Cada um dos itens da lista apresentada anteriormente é uma expressão de busca com termos a serem submetidos ao Twitter, que por sua vez deve retornar postagens referentes a esses termos. Por exemplo, o primeiro (1) item busca *tweets* que obrigatóriamente contém em seu texto a palavra segurança (*security*) junto com um ou mais dos seguintes termos: vírus (*virus*), *worm*, ataque (*attack*), intrusão (*intrusion*), invasão (*invasion*), DDoS (ddos), *hacker* ou *cracker*. Ou seja, a primeira expressão busca por *tweets* relacionados com ataques contra redes de computadores. Analisando então cada item anterior: o segundo (2) busca por *tweets* que relatam vulnerabilidades, principalmente em softwares; o terceiro (3) busca por problemas de segurança de dia-zero; o quarto (4) por mensagens que informem a respeito do uso de softwares maliciosos que controlam máquinas remotamente; o quinto (5) por problemas de segurança em algumas tecnologias muito utilizadas; o

sexto (6) busca especificamente por *spyware*; o sétimo (7) por falhas em softwares; e o oitavo (8) por problemas bem específicos em elementos da camada de aplicação ou superior. Esses são exemplos de termos de buscas criados durante as pesquisas Campiolo *et al.* (2013); Santos *et al.* (2012, 2013), contudo outros termos e expressões podem ser acrescentados visando obter informações atuais e que ajudem a evidenciar problemas em ambientes computacionais.

Neste trabalho a submissão das expressões e termos de busca ao Twitter é realizada através do uso da API do Twitter, que pode ser acessada utilizando-se bibliotecas de terceiros (Twitter4J, 2016) ou do próprio Twitter (Gnip, 2016). Portanto, através dessas bibliotecas é possível desenvolver softwares que permitam coletar mensagens postadas no Twitter. Durante o desenvolvimento deste trabalho foram desenvolvidos dois softwares coletores, para testes, sendo um utilizando a biblioteca Gnip (Gnip, 2016)[8] e outro a biblioteca Twitter4J (Twitter4J, 2016) – os dois softwares desempenham a mesma função, o que muda é a biblioteca de acesso ao Twitter. Ambas bibliotecas testadas apresentaram resultados muito similares no tocante ao número de *tweets* recuperados nas buscas utilizando termos de segurança de computadores. Devido a isso optou-se por utilizar a biblioteca Twitter4J que possui licença Apache 2.0 (Apache, 2016c) e não requer pagamento pelo acesso aos dados do Twitter.

Na teoria, as bibliotecas que dão acesso a API do Twitter permitem recuperar *tweets* de alguns dias passados, mas na prática não é incomum conseguir recuperar apenas *tweets* de algumas horas atrás. Então, para este trabalho o software coletor desenvolvido utiliza a estratégia de realizar buscas constantes por *tweets* de segurança em intervalos regulares de 1 (um) minuto, ou seja, a cada minuto o software contata o Twitter para verificar se há mensagens relacionadas com os termos de busca. Caso o software coletor retorne *tweets*, esses deverão ser salvos em uma base de dados. Contudo, como a busca é constante e em um curto intervalo de tempo, não é incomum recuperar *tweets* que já foram salvos na base de dados. Desta forma, o software coletor e a base de dados devem descartar os *tweets* duplicados. Essa operação é possível, pois cada *tweet* possui um número de identificação único, retornado pela API.

Mesmo usando a abordagem de buscar por *tweets* postados a cada minuto, não há como garantir que todos os *tweets* postados serão recuperados, mas no geral os *tweets* retornados representam uma boa amostragem do que os usuários estão escrevendo a respeito de problemas computacionais nas redes sociais (Campiolo *et al.*, 2013; Santos *et al.*, 2012, 2013).

4.2.5.2 Remover *tweets* indesejados

Mesmos utilizando termos de buscas direcionados à área de segurança de computadores não é incomum que sejam capturados e armazenados na base de dados *tweets* que noticiam assuntos de outros contextos. Também, dentre os *tweets* coletados existem casos de *spam* e *tweets* mal formados, ou seja, que não trazem informações úteis e portando devem ser removidos. Desta maneira, é função do presente passo filtrar e remover essas mensagens para evitar problemas com falsos positivos. A Figura 4.19 apresenta o fluxograma do método utilizado para remover *tweets* indesejados.

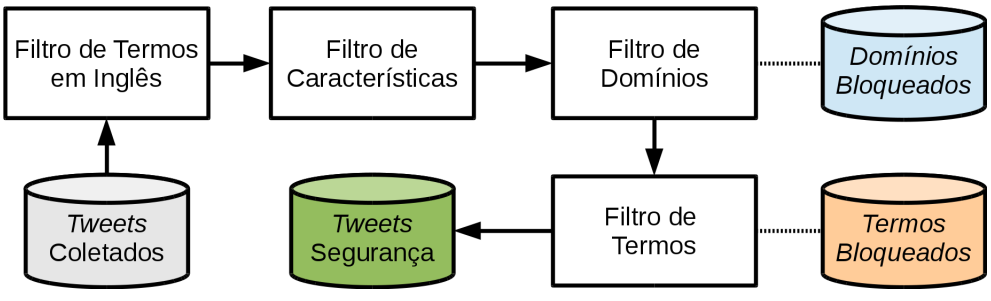


Figura 4.19: *Fluxograma do processo de filtragem (Santos et al., 2013).*

O processo de filtragem inicia com a análise de todos os *tweets* coletados. Assim, os *tweets*

[8] Agradecemos a Gnip (2016) pelo acesso concedido gentilmente para testes.

coletados no passo anterior são submetidos ao Filtro de Termos em Inglês, que deve remover mensagens que não são da língua inglesa. Nesse trabalho foi adotado por padrão só analisar mensagens em inglês, pois em Santos *et al.* (2012) notou-se que mensagens em português não traziam muitas informações relevantes a respeito de ameaças computacionais, o que felizmente não é verdade nos *tweets* em inglês. Contudo, mesmo utilizando termos de busca em inglês não é incomum que sejam retornados *tweets* em outras linguás, por isso, se faz necessário essa filtragem. Na prática a exclusão dos *tweets* em outros idiomas é possível, principalmente através da análise de um campo que indica o código da linguagem do texto, que é retornado junto com o *tweet* através da API do Twitter.

Na sequencia os *tweets* são submetidos ao Filtro de Características, que tem por função remover mensagens que possuem as seguintes características:

- Menos de 4 (quatro) palavras;
- Tamanho inferior a 40 (quarenta) caracteres;
- Mais de 3 (três) URL (*Uniform Resource Locator*);
- Número de *hashtags* superior a metade do número de termos da mensagem;
- Número de menções a usuários superior a metade do número de termos da mensagem.

Neste trabalho, os *tweets* que possuem as características citadas anteriormente são chamados de mal formados. Tais *tweets* são excluídos do processamento, pois em Campiolo *et al.* (2013); Santos *et al.* (2012, 2013) notou-se que os *tweets* mal formados, na maioria dos casos não apresentam informações uteis e podem atrapalhar posteriormente, no processo de agrupamento de mensagens.

O Filtro de Domínios remove *tweets* irrelevantes como alertas de segurança para computadores, considerando o nome de domínio usado na Internet (Tanenbaum e Wetherall, 2013) e que está presente nas URL contidas no corpo do texto de muitos *tweets*. Em Santos *et al.* (2013) notou-se que cerca de 84,9% dos *tweets* coletados fazem referências a páginas Web, por intermédio de URL no corpo da mensagem. Isso ocorre porque o texto do *tweet* só comporta 140 caracteres, o que pode ser pouco para detalhar um problema de segurança. Devido a esse fato, tornou-se corriqueiro que o retador do *tweet* faça uso de referências (*links*) para páginas Web, que detalham/expandem melhor o assunto do *tweet*. Desta forma, o Filtro de Domínios evita mensagens fora de contexto comparando o domínio contido na URL do *tweet*, com uma lista negra (*blacklist*) de domínios que conhecidamente não noticiam a respeito de assuntos ligados a segurança de computadores.

A lista negra de domínios é representada na Figura 4.19 pela base de dados Domínios Bloqueados. Essa lista negra foi feita em Santos *et al.* (2013), através da extração e análise manual de URL de milhares de *tweets*. Atualmente a lista possui 161 domínios a serem bloqueados, mas essa base de dados pode ser atualizada de tempos em tempos. A seguir são apresentados alguns dos domínios que fazem parte da lista negra de domínios:

1. washingtonpost.com
2. forbes.com
3. bbc.co.uk
4. cnn.com
5. foxnews.com
6. aljazeera.com
7. nbcnews.com
8. israelnationalnews.com
9. uol.com.br

Através dos domínios apresentados anteriormente é possível observar que muitos são domínios de páginas Web de grandes empresas de comunicação, cujo o foco dessas não é noticiar a respeito de informática. Assim, *tweets* de segurança relacionados com esses domínios, normalmente citam problemas de segurança tal como, guerras entre diferentes etnias. Ou seja, comumente esses *tweets* não estão relacionados com segurança de computadores e se não forem removidos causarão falsos positivos.

Ainda no Filtro de Domínios, cada URL de *tweet* antes de ser submetida a lista negra de domínios, precisa passar por um processo de conversão de URL curta para longa. Isso é necessário,

pois para economizar espaço no texto, os *tweets* utilizam um esquema de URL curta, que não informa o domínio original. Por exemplo, a URL curta:

- https://t.co/MfEoYX4gCy

quanto expandida se torna a URL:

- http://securityaffairs.co/wordpress/44074/hacking/openssl-flaw.html?utm_source=dlvr.it&utm_medium=twitter

Assim, para se conhecer o nome do domínio de uma URL curta é necessário convertê-la para o seu formato original (longo). Finalmente, de posse da URL longa o Filtro de Domínios compara a URL do *tweet* os domínios da lista negra. Caso a URL do *tweet* combine com alguma URL da lista esse é excluído.

Outro problema presente nos *tweets* coletados são as mensagens de *spam*. Algumas mensagens capturadas são, por exemplo, propagandas de antivírus. Esse tipo de *tweet* é capturado, pois o seu texto normalmente está de acordo com os termos de busca empregados na captura de *tweets* que relatam problemas de segurança em computadores. Desta forma, é tarefa do Filtro de Termos excluir tanto os *tweets* que contenham textos referentes a *spam* como mensagens fora do contexto de segurança de computadores. Para isso, cada palavra do texto do *tweet* é comparada a uma lista negra composta de termos que comumente estão presentes em mensagens de *spam* ou fora do contexto de segurança de computadores. Caso os termos do *tweet* combinem com algum termo da lista, esse é excluído.

A lista negra de palavras relacionadas com *spam* e mensagens fora de contexto é representada na Figura 4.19 pela base de dados Termos Bloqueados. Similarmente a base Domínios Bloqueados, os termos presentes na base Termos Bloqueados foram criados durante a pesquisa Santos *et al.* (2013), que analisou milhares de *tweets* para gerar uma lista contento palavras que frequentemente simbolizam *spam* e outras mensagens irrelevantes. Atualmente a lista contém 119 termos, sendo alguns:

1. *antivirus*
2. *free*
3. *murder*
4. *kill*
5. *gunmen*
6. *bomb*
7. *soldiers*
8. *obama*
9. *nuclear*

Conforme é possível observar nos itens apresentados anteriormente, a lista de *spam* é composta por palavras relacionadas com *spam*, tal como antivírus (*antivirus*) e *free* (normalmente uma referência para gratuito, ou seja, sem custos). Contudo, muitas dessas palavras também remetem a mensagens fora do contexto desejado por este trabalho. Por exemplo, não é nada comum que mensagens relacionadas com problemas em computadores utilizem palavras como assassinato (*murder*), matar (*kill*) e atirador (*gunmen*), esse tipo de *tweet* normalmente relata algum tipo de ameaça fora do mundo virtual. Já, palavras como bomba (*bomb*), soldados (*soldiers*), obama (referência ao presidente dos Estados Unidos da América, Barack Obama) e nuclear, são normalmente utilizados para relatar ameaças à segurança de nações e não estão diretamente ligadas ao contexto de informática. Ou seja, *tweets* contendo esses termos devem ser removidos para não influenciar negativamente no processo de geração de alertas a respeito de ameaças contra computadores.

No final desse passo, aquele *tweet* que não foi removido por nenhum dos filtros apresentados anteriormente, será salvo na base de dados *Tweets* Segurança (Figura 4.19) e servirá como fonte de informação para a geração de alertas de segurança a respeito de problemas com a segurança de redes de computadores.

4.2.5.3 Agrupar *tweets* com assuntos similares

Depois de filtrar os *tweets* coletados restará em sua maioria, mensagens que relatam problemas de segurança em computadores. Todavia, dependendo da quantidade de *tweets* pode ser inviável

para um administrador ou mesmo para um sistema computacional analisar o texto de centenas ou milhares de *tweets*, para só então tomar ciência dos principais problemas de segurança que estão atualmente afetando as redes de computadores mundo a fora. Então, o passo Agrupar por Similaridade (Figura 4.18) propõem agrupar *tweets* levando em conta a similaridade de seus textos/assuntos. Dessa maneira, é possível reduzir a quantidade de informação a ser analisada posteriormente, em um processo de identificação de problemas, bem como ajudar a evidenciar quais são as principais ameaças contra a segurança de computadores que estão sendo discutidas no Twitter.

Neste trabalho utiliza-se o Apache Lucene (Apache, 2016b) para realizar o agrupamento de *tweets* semelhantes. O Apache Lucene é uma biblioteca que permite indexar textos para posteriormente recuperá-los, utilizando para isso termos de busca. É um processo bem similar ao empregado em motores de busca que recuperam conteúdo da Internet, tal como o Google (https://www.google.com/). Ou seja, o Apache Lucene permite realizar buscas a respeito do conteúdo dos textos indexados em seu motor de busca. O processo de consulta se dá através da submissão de termos de busca e o resultado é uma lista, na qual cada possível elemento retornado é composto de dois valores, sendo estes:

1. Texto indexado e que corresponde ao termo de busca utilizado;

2. Escore de similaridade, que é um número que indica o quão semelhante é o texto encontrado, frente ao termo de busca utilizado.

O Apache Lucene emprega um cálculo de similaridade de cosseno simplificado (Manning *et al.* , 2008) que também usa variáveis ajustáveis para calcular o escore de similaridade, sendo então possível atribuir pesos para diferentes termos da consulta, amenizar parágrafos repetidos no texto, entre outros (Apache, 2016b). No esquema de escore de similaridade do Apache Lucene, quanto maior for o número retornado, mais similar é o texto com o termo de busca.

A estratégia proposta para se agrupar os *tweets* e evidenciar mensagens sobre segurança de computadores é indexar os *tweets* da base de dados *Tweets* de Segurança (Figura 4.19) e posteriormente comparar cada *tweet* da base com os demais *tweets*. Portanto, o termo de busca a ser submetido ao Apache Lucene é um texto de *tweet*. Ou seja, após os *tweets* da base de dados *Tweets* Segurança serem indexados no Apache Lucene o processo de agrupamento inicia com a submissão de um *tweet* qualquer da base de dados *Tweets* Segurança ao motor de busca do Apache Lucene e o resultado será um grupo com os *tweets* similares ao *tweet* que originou a consulta – um *tweet* só é adicionado ao grupo se o seu escore de similaridade retornado atingir um determinado valor (que será detalhado a seguir). Com o grupo formado, todos os *tweets* desse grupo são removidos da base *Tweets* Segurança. E o processo continua até que não haja mais *tweets* na base de dados *Tweets* Segurança. É importante notar que um *tweet* só pode fazer parte de um grupo, ou seja, um *tweet* não pode estar em mais de um grupo. Isso é feito para evitar que se tenha vários grupos com os mesmos *tweets*.

Durante o processo de agrupamento utiliza-se o valor do escore de similaridade para decidir se os *tweets* são semelhantes e se devem ser agrupados. Assim, deve-se comparar o escore de similaridade retornado com um valor que representa o limiar entre os textos dos *tweets* serem considerados semelhantes ou não. Neste trabalho esse valor limiar leva o nome de grau de similaridade. O valor do grau de similaridade utilizado é muito importante, pois influência diretamente na qualidade dos grupos gerados. O grau de similaridade deve ser escolhido de forma que cada grupo formado contenha *tweets* que relatem exclusivamente um único assunto e que um mesmo assunto não esteja espalhado em vários grupos. Há dois extremos de escolha de grau de similaridade que devem ser evitados:

1. Se o valor for muito baixo, provavelmente serão criados poucos grupos, sendo que cada grupo pode ser composto de *tweets* que relatam problemas distintos. Esse tipo de agrupamento não é desejável, já que alguns *tweets* seriam agrupados erroneamente fora de seu contexto;

2. Se o valor for muito alto, exite a tendencia de serem criados muitos grupos e que grupos

distintos relatem o mesmo assunto. Essa situação também não é a ideal, pois esperá-se que *tweets* que relatem o mesmo problema de segurança estejam agrupados em um único lugar.

Ou seja, deve-se utilizar um grau de similaridade médio, de forma que não haja vários grupos relatando o mesmo assunto e também não deve haver *tweets* abordando assuntos diferentes em um mesmo grupo. O ideal seria que *tweets* relatando um dado problema de segurança estejam em um grupo diferente dos *tweets* que relatam a solução desse mesmo problema.

Em Santos *et al.* (2012) chegou-se a conclusão que um bom grau de similaridade para se agrupar *tweets* seria algo em torno de 0,5 ou 0,75, pois esses valores normalmente são retornados pelo Apache Lucene, quando se comparam *tweets* muito semelhantes ou até mesmo *retweets*. Contudo em experimentos posteriores (Campiolo *et al.*, 2013; Santos *et al.*, 2012), utilizando uma quantidade maior de *tweets*, descobriu-se que usar um grau de similaridade fixo pode produzir resultados ruins devido ao tamanho variável dos textos dos *tweets*. O problema ocorre quando se submete mensagens pequenas (menos de 70 caracteres) como termos de busca. Nesse caso, há uma grande chance das poucas palavras contidas nesse *tweet* de texto curto combinar com as muitas palavras dos *tweets* com textos mais longos, o que retorna falsamente um alto grau de similaridade e por consequência acaba agrupando vários *tweets* com contextos diferentes. Para resolver esse problema propõem-se o uso de um grau de similaridade que varie de acordo com o tamanho do texto do *tweet* que será utilizado como termo de busca. O calculo desse grau de similaridade dinâmico é dado pela Equação 4.1, na qual:

$$GrauSimilaridadeExigido = ((\delta - \frac{x * \delta}{160}) * \alpha) + \delta \tag{4.1}$$

- δ é o grau de similaridade mínima aplicada a uma mensagem de tamanho máximo. Pode ser por exemplo, 0,5 ou 0,75;
- 160 é o tamanho máximo em caracteres de uma mensagem do Twitter. Em Campiolo *et al.* (2013); Santos *et al.* (2013) observou-se que a API do Twitter pode retornar mensagens com até 160 caracteres. Por isso, neste trabalho considera-se o tamanho máximo de um *tweet* como sendo de 160 caracteres e não 140;
- x é o número de caracteres da mensagem que será utilizada como termo de busca na análise de agrupamento;
- α é um fator extra de crescimento aplicado ao grau de similaridade dinâmico, calculado em função do número de caracteres presentes no *tweet* usado como termo de busca. Esse pode ser por exemplo 2 ou 8. A influência desse valor pode ser observado na Figura 4.20.

A Figura 4.20 apresenta o comportamento da Equação 4.1 utilizando dois valores de α diferentes (2 e 8), um δ igual a 0,75 e variando x entre 1 e 160. No exemplo da Figura 4.20, com o α valendo 2 (linha azul com círculos) ao se utilizar um *tweet* de 160 caracteres como termo de busca serão considerados similares os *tweets* retornados pelo Apache Lucene, com um escore de similaridade igual ou superior a 0,75, já que esse é o grau de similaridade exigido. No mesmo caso se for um *tweet* de 80 caracteres, o grau de similaridade exigido para que um *tweet* faça parte do agrupamento aumenta para 1,5. Já se for um *tweet* de 40 caracteres com α igual a 2, será exigido um grau de 1,88. No caso do α com valor 8 (linha vermelha com asterisco) um *tweet* com 160 também exige um grau de similaridade de 0,75, mas esse valor sobe para 3,75 e 5,25 para *tweets* de busca com respectivamente 80 e 40 caracteres.

Então, na Equação 4.1 quanto menor o texto do *tweet* utilizado como termo de busca, mais alto será o grau de similaridade exigido para gerar o grupo. Isso evita que *tweets* pequenos influenciem negativamente no resultado final do agrupamento das mensagens de segurança postadas no Twitter.

Após agrupar todos os *tweets* da base de dados *Tweets* Segurança o resultado é uma lista contendo os grupos formados. Cada grupo dessa lista é composto por (1) uma mensagem/texto que representa a informação desse grupo e (2) os *tweets* que formam o grupo. A mensagem que representa a informação do grupo é o texto do *tweet* que foi utilizado como termo de busca para

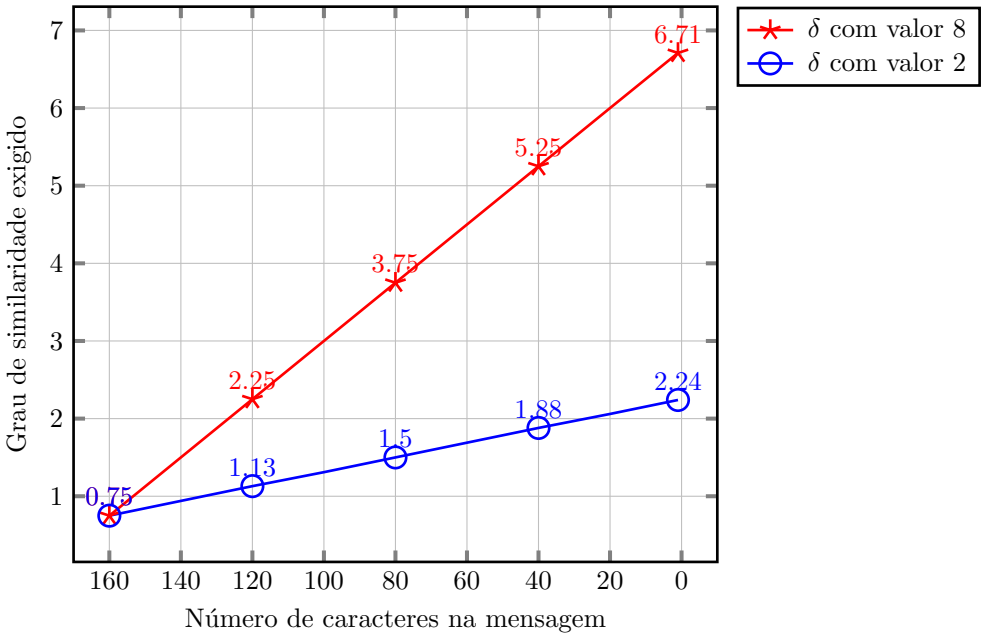


Figura 4.20: *Comportamento do grau de similaridade dinâmico exigido segundo o tamanho do texto do tweet para dois fatores de crescimento (δ) distintos.*

formar o grupo. Um grupo pode conter apenas um único *tweet*, caso o *tweet* usado como termo de busca não retorne nenhum outro *tweet* que atenda o grau de similaridade exigido.

4.2.5.4 Gerar lista de grupos de *tweets* relevantes

Mesmo depois de agrupar os *tweets* que relatam problemas de segurança é possível que ainda restem centenas ou milhares de mensagens que representam os grupos formados, ou seja, ainda pode ser árdua a tarefa de ler e analisar todas essas mensagens para descobrir quais são relevantes como alertas de segurança para redes de computadores.

Um fator muito comum e importante na evidenciação de mensagens relevantes postadas em redes sociais é o número de vezes que uma dada informação foi replicada. No caso do Twitter é possível constatar a relevância de uma informação observando, por exemplo, o número de *retweets* (Morris *et al.*, 2012). Desta maneira, o número de *tweets* similares em um grupo é fundamental para determinar se uma mensagem é relevante ou não.

Contudo durante experimentos realizados em Santos *et al.* (2013), observou-se que o principal fator para determinar a relevância da mensagem de um grupo, não é o número de *tweets* similares, mas sim o número de usuários do Twitter, que enviaram essas mensagens. Isso por que mensagens de *spam* costumam ter muitos *tweets*, contudo esses *tweets* são enviados por poucos usuários. Já grupos que relatam acontecimentos importantes, tal como um problema de segurança que pode afetar muitos computadores, são formados por muitos *tweets* que são enviados da mesma forma por muitos usuários. Assim, neste trabalho considera-se a informação de um grupo como relevante, apenas quando os *tweets* desse grupo forem postados por no mínimo 10 usuários distintos. Pois, nesse caso pelo menos 10 pessoas acharam a informação relevante o suficiente para compartilhá-la.

Além do número de *tweets* e usuários que compõem o grupo também é possível analisar outras características dos *tweets* que formam um grupo para aumentar ou diminuir sua relevância. Tal como:

Frequência das palavras: É possível analisar palavra por palavra da mensagem do *tweet* que formou o grupo e verificar se alguma dessas palavras teve o seu uso mais intensificado nos últimos dias ou horas. Caso haja alguma palavra nessas condições, isso pode indicar que a mensagem trata de um assunto em evidência. Isso é muito semelhante ao conceito de *trend topics* utilizado no Twitter;

Ocorrência de palavras raras: Essa técnica busca por palavras raras que podem ocorrer nas mensagens que formam o grupo. A ocorrência de palavras raras ou novas pode indicar o surgimento de algum assunto novo, tal como uma ameaça dia-zero. Portanto, esse tipo de acontecimento aumenta a relevância da mensagem. Todavia durante testes (Santos *et al.* , 2013) notou-se que se uma mensagem apresentar mais que duas palavras raras, essa tende a ser uma mensagem fora de contexto e por isso deve ter o seu grau de relevância reduzida;

Picos de *tweets* e tempo de propagação: No geral os eventos relatados pelos usuários do Twitter têm um padrão de propagação, principalmente nas notícias de última hora, tal como *tweets* a respeito de catástrofes (Phuvipadawat e Murata, 2010; Qu *et al.*, 2011). Normalmente, para esse tipo de notícia são postados alguns *tweets* no primeiro dia, no segundo dia há um grande pico de postagens (uma grande elevação no número de *tweets* a respeito do assunto) e conforme os dias passarem a quantidade de *tweets* diminui gradativamente, até que o assunto não seja mais comentado. Os *tweets* que relatam problemas na segurança de computadores seguem o mesmo padrão (Santos *et al.*, 2013), ou seja, no segundo dia normalmente há uma explosão no número de *tweets* postados e no período de 10 a 14 dias o assunto deixa de ser comentado no Twitter. Desta forma, quando um grupo de *tweets* similares se enquadram nessas características aumenta-se o grau de relevância. Em contrapartida, *tweets* a respeito de *spam* não possuem dia de pico e costumam ser propagados por muito mais tempo, assim se um grupo reagir dessa forma, diminui-se seu grau de relevância.

Então, o fator inicial para se considerar a relevância de uma dada mensagem de um grupo de *tweets* semelhantes é o número de usuários que postou esses *tweets*. A esse número dá-se o nome de grau de relevância. Posteriormente, esse grau de relevância pode ser ainda aumentado ou diminuído analisando-se características como frequência das palavras, ocorrência de palavras raras, picos de *tweets* e tempo de propagação. Dependendo o caso, a cada um desses fatores pode-se acrescentar ou diminuir, por exemplo, 10% do grau de relevância atual do grupo.

O Algoritmo 4 resume toda a metodologia de agrupamento descrita na Subseção 4.2.5.3, bem como a geração da lista de *tweets* relevantes apresentada nesta subseção. O algoritmo inicia indexando os dados da base *Tweets* Segurança (linha 1). O fator de crescimento (α) e o grau de similaridade mínima (δ) são configurados respectivamente em 2 e 0,75 (linhas 2 e 3). Depois inicia o processo de comparação para agrupamento dos *tweets* de segurança, que deve ser realizado enquanto houver *tweets* de segurança na lista (entre a linha 4 e 17). O processo de agrupamento começa obtendo-se um *tweet* qualquer, adicionando-o na lista que formará o grupo de *tweets* similares, extraindo o número de caracteres desse *tweet* e o grau de similaridade que será exigido durante a comparação desse com os demais *tweets* (respectivamente linhas 5, 6, 7 e 8). Na linha 9, o texto do *tweet* selecionado é submetido como termo de busca no Apache Lucene que retorna os *tweets* correspondentes e os respectivos escores de similaridade. Na sequência, agrupa-se apenas os *tweets* retornados que possuem um escore de similaridade maior ou igual ao grau de similaridade exigido (linhas 10 a 12). Depois, verifica-se o grau de relevância do grupo recém formado, o que é feito extraindo-se o número de usuários distintos, que fazem parte do grupo (linha 13). Se esse número for maior ou igual a 10 (linha 14), então é calculado o grau de relevância do grupo (linha 15), grava-se em uma lista a mensagem do grupo recém criado e o grau de relevância desse (linha 16). Em seguida, os *tweets* do grupo são removidos da lista de *tweets* de segurança (linha 17). No fim, depois de processar todos os *tweets* de segurança, o algoritmo irá retornar uma lista com as mensagens dos grupos mais relevantes encontrados durante o processo de agrupamento.

A lista com as mensagens a respeito de problemas de segurança postados no Twitter extraída do Algoritmo 4 é salva na base de dados Ameaças Citadas em Redes Sociais do módulo Base de Conhecimento (Figura 4.3) e fica disponível para o restante da arquitetura proposta. De posse das principais notícias a respeito de segurança de computadores postadas na Internet é possível utilizá-las para:

- Manter o administrador da rede informado a respeito dos principais problemas que estão surgindo e podem afetar a rede local. Assim, medidas preventivas podem ser tomadas pelo

Algoritmo 4: Agrupamento por similaridade (Santos *et al.*, 2013)

Entrada: Um vetor de *tweets* - *tweetsSegurana*
Saída: Mensagens consideradas importantes por serem postadas por mais de 10 usuários

```
1  ApacheLucene.indexa(tweetsSegurana)
2  α ← 2
3  δ ← 0,75
4  enquanto tweetsSegurana ≠ φ faça
5      tweet ← tweetsSegurana.proximo()
6      tweetsSimilares.adiciona(tweet)
7      x ← tweet.textoMensagem.tamanho()
8      grauSimilaridadeExigido ← ((δ − x∗δ/160) ∗ α) + δ
9      tweetsLucene ← ApacheLucene.obterTweetsSimilares(tweet.textoMensagem)
10     para cada tweetsLucene.proximo() faça
11         se tweetsLucene.score ≥ grauSimilaridadeExigido então
12             tweetsSimilares.adiciona(tweetsLucene.tweet)
13     numeroUsuarios ← removerTweetsComUsuariosRepetidos(tweetsSimilares).tamanho()
14     se numeroUsuarios ≥ 10 então
15         grauRelevancia ← calculaRelevanciaGrupo(numeroUsuarios, tweetsSimilares)
16         mensagensImportantes.adiciona(grauRelevancia,tweet.textoMensagem)
17     tweetsSegurana.remover(tweetsSimilares)
18 retorna mensagensImportantes
```

administrador para evitar vulnerabilidades que estão sendo exploradas por *hackers*. Essas informações são disponibilizadas ao administrador através de uma interface (ver Figura 4.1);

- Correlacionar os alertas da base de dados Ameaças Citadas em Redes Sociais com as informações a respeito da rede local (representados pelas bases Alertas de Segurança e Dados de Uso da Rede do módulo Base de Conhecimento) para auxiliar na identificação e correção de problemas que afetam a rede local. Essa correlação entre os alertas da rede local e das redes sociais pode ser feita manualmente pelo administrador da rede, utilizando para isso a interface proposta pela arquitetura, contudo técnicas de correlação de dados também podem ser empregadas para relacionar automaticamente tais alertas.

Portanto, este trabalho propõem o uso de uma nova[9] e rica fonte de informações a respeito de problemas de segurança em computadores, que são as redes sociais. Devido ao uso dessa nova fonte de informações utiliza-se neste trabalho, os termos: (1) **alertas de segurança de alto nível** para os alertas de segurança gerados a partir de informações postadas em redes sociais e (2) **alertas de segurança de baixo nível** para as fontes de informações mais tradicionais como IDS, registros do sistema (*logs*), etc. Por fim, espera-se que os alertas de alto nível melhorem o nível de percepção do administrador a respeito dos riscos que podem afetar a rede, bem como permitam relacioná-los aos alertas de baixo nível para então diminuir a incidência de problemas de segurança que afetam as redes de computadores locais.

[9] Nova no sentido de pouco explorada para a geração de alertas a respeito de ameaças contra a segurança de computadores.

Capítulo 5

Experimentos e resultados

Este capítulo apresenta os experimentos realizados com a arquitetura autonômica proposta e com o uso das redes sociais como fonte para geração de alertas a respeito de ameaças contra ambientes computacionais. Por fim, é feita uma análise dos resultados obtidos com os experimentos.

5.1 Experimentos

Esta seção apresenta os experimentos realizados para avaliar a metodologia proposta por este trabalho. São descritos os cenários de cada experimentos, objetivos e os dados obtidos. A Subseção 5.1.1 apresenta os experimentos com o Of-IDPS e a Subseção 5.1.2 os experimentos com a extração de alertas de segurança do Twitter.

5.1.1 Experimentos com o Of-IDPS

Os experimentos com o Of-IDPS foram realizados em um cenário composto por duas redes: interna e externa (Figura 5.1). A rede interna é formada por quatro computadores que podem assumir o papel de cliente ou servidor, um IDS, um controlador OpenFlow e um *switch* OpenFlow. A rede externa possui um *switch* comum e dois computadores que também podem ser clientes e servidores. As redes interna e externa estão ligadas através de um roteador.

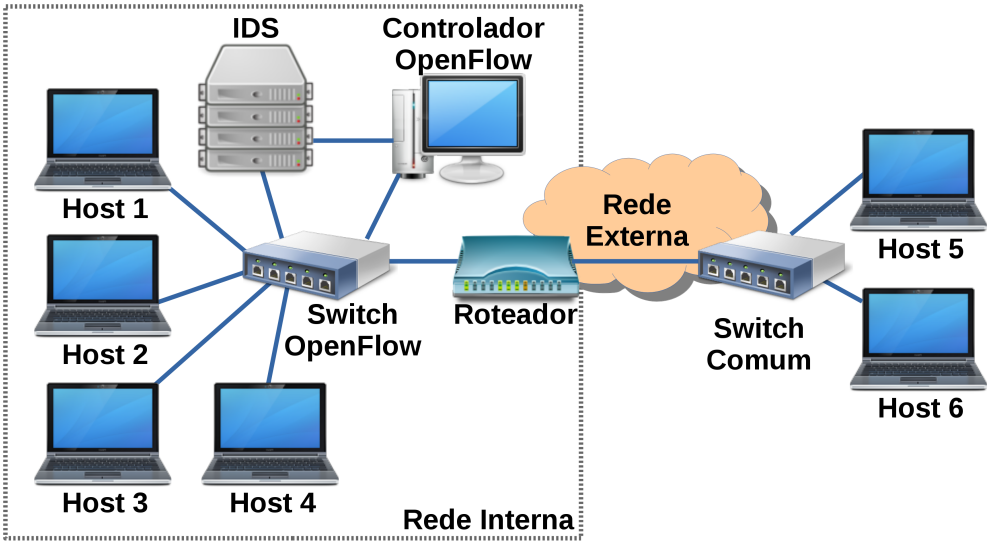


Figura 5.1: *Cenário de rede utilizado para analisar a efetividade e limitações do Of-IDPS.*

Os elementos do cenário são simulados usando o Mininet 2.1.0, exceto o controlador. Nos *hosts* simulados é executado o sistema operacional Linux com *kernel* 3.8.0-19 e a distribuição Ubuntu

13.04. O controlador OpenFlow é executado na mesma máquina hospedeira dos elementos simulados pelo Mininet. A configuração do hospedeiro é um Intel Core i5-3337U de 1.80 GHz (*Gigahertz*), com 6 GB (*Gigabyte*) de memória principal. O sistema operacional é um Linux *kernel* 3.13.3 com a distribuição Slackware Linux 14.0. O Of-IDPS foi desenvolvido utilizando o arcabouço de controlador OpenFlow Beacon com a versão 1.0.4. Durante os experimentos o controlador executa o código do Of-IDPS e também o código de um controlador Beacon OpenFlow simples (sem Of-IDPS), para futuras comparações. O Beacon foi escolhido por apresentar melhor desempenho se comparado com outros controladores OpenFlow (Erickson, 2013).

O *switch* OpenFlow executa o Open vSwitch versão 1.9.0. Cada porta do vSwitch é configurada para suportar três configurações de largura de banda, sendo: uma que utiliza toda largura de banda disponível; outra com uma restrição suave; e a última com restrição severa. Esse *switch* também foi configurado para espelhar todo seu tráfego para a porta do IDS. Assim, o IDS pode analisar todos os pacotes que passem pela rede e gerar alertas de segurança que serão utilizados pelo Of-IDPS.

A API em Python do simulador Mininet foi utilizada para prover experimentos mais confiáveis e uniformes, pois possibilita programar os cenários dos experimentos. Dessa forma, é possível controlar a temporização e ordem na execução de comandos, facilitando a recriação dos experimentos apresentados neste trabalho. Diferentemente de trabalhos similares, o código-fonte do Of-IDPS, os *scripts* de configuração do *switch* OpenFlow (controle de banda e espelhamento de porta) e o programa que automatiza a execução dos experimentos estão disponibilizados publicamente em https://github.com/luizsantos/Of-IDPS. Dessa forma outros pesquisadores podem utilizar o Of-IDPS nos seus projetos ou confrontar/validar os resultados relatados por este trabalho.

5.1.1.1 Experimento 1 - Efetividade e resposta do Of-IDPS a alertas do IDS

Neste experimento é analisada a viabilidade e efetividade da ferramenta, ou seja, se o tráfego de rede malicioso é regulado sem influenciar negativamente os outros fluxos da rede. Também é analisado o tempo de resposta desde o envio do alerta até a reação do Of-IDPS frente ao problema de segurança detectado.

O experimento começa com dois fluxos de rede em paralelo e ambos são considerados normais pelo sistema de detecção. Pouco tempo depois um dos fluxos é considerado suspeito e são emitidos alertas de segurança passando respectivamente pelos níveis de prioridade de segurança baixo, médio e alto. Para executar esse teste foi utilizado um servidor Iperf (Barayuga e Yu, 2015; Iperf, 2016; Olvera-Irigoyen *et al.*, 2011) que executa serviços nas portas TCP 80 e 90. Dois *hosts* realizam acessos simultâneos ao servidor usando o cliente Iperf (o Iperf é uma ferramenta que tem por objetivo medir a largura de banda da rede. Então, por padrão o Iperf irá tentar transmitir o maior número de pacotes/dados possível na rede, visando verificar a velocidade da transmissão (Iperf , 2016)). O acesso do *host* que acessa a conexão TCP/80 é considerado normal e o acesso do *host* na porta 90 é considerado malicioso e são emitidos alertas para esse fluxo da seguinte forma: alerta de risco baixo após 30 segundos, alerta de risco médio após 60 segundos e alerta de risco alto após 90 segundos. Assim, espera-se observar a execução e efetividade do Of-IDPS considerando quatro situações distintas: tráfego normal e três níveis de alertas de segurança.

O experimento foi executado 30 vezes e a média dos resultados é calculada considerando o tráfego recebido pelo servidor, sendo essa apresentada na Figura 5.2. O "Fluxo Legítimo" representa o tráfego TCP na porta 80, o "Fluxo Malicioso" representa o tráfego TCP na porta 90, "Alerta Nível Baixo", "Alerta Nível Médio" e "Alerta Nível Alto" indica respectivamente o momento em que cada nível de alerta foi emitido do IDS para o controlador Of-IDPS. A Figura 5.3 apresenta o desvio padrão do fluxo legítimo/normal e malicioso dos 30 testes.

No gráfico da Figura 5.2 é observado que durante os primeiros 30 segundos os fluxos das portas 80 e 90 disputam a largura de banda da rede para acessar o servidor. Durante esse primeiro momento a taxa de transferência de dados na porta 80 foi de 6,47 Mbps e na porta 90 foi de 9,87 Mbps (Tabela 5.1). Depois de 30 segundos é emitido o alerta de nível baixo e com 33 segundos é aplicada a restrição de largura de banda ao fluxo da porta 90. Com 39 segundos o fluxo é estabilizado com a restrição de largura de banda sendo aplicada. Durante esse período a taxa de transferência foi

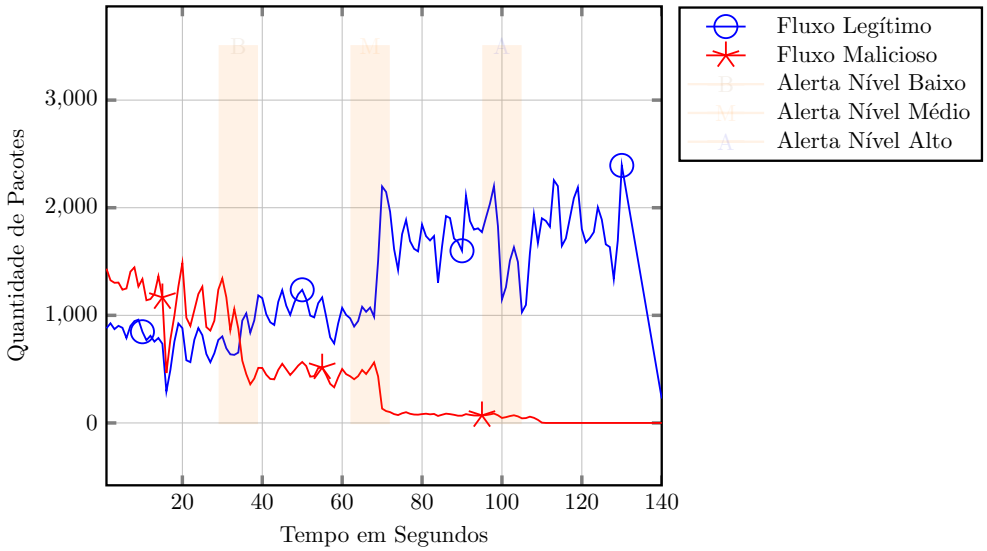


Figura 5.2: *Experimento 1 - Testes com controle de largura de banda e bloqueio de pacotes com Of-IDPS.*

de 8,88 Mbps para a porta 80 e 3,78 para a porta 90. Aos 64 segundos é emitido o alerta de nível médio e a restrição de largura de banda severa é aplicada a 70 segundos, o fluxo é estabilizado a 72 segundos. Nesse período o fluxo malicioso (porta 90) teve sua taxa de transferência de dados alterada para 0,56 Mbps, enquanto o fluxo legítimo (porta 80) aumentou para 14,72 Mbps. A 96 segundos é emitido o alerta de nível alto, logo o fluxo malicioso é bloqueado, ocorrendo a detecção a 108 segundos e o bloqueio a 110 segundos, então a taxa de transferência de dados para o fluxo legítimo atinge 15,04 Mbps. Assim, o Of-IDPS consegue regularizar/padronizar os fluxos maliciosos o que também pode ser constatado observando o desvio padrão do fluxo suspeito, apresentado na Figura 5.3, no qual o desvio padrão torna-se mais estreito e regular após cada emissão de alerta de segurança ao fluxo considerado malicioso, o que demostra a efetividade do controle de limitações de recursos impostos pelo Of-IDPS.

Tabela 5.1: *Experimento 1 - Taxa média de transferência de dados dos fluxos durante experimento.*

Restrição Aplicada/Período de tempo em segundos	Normal/1-30	Leve/39-64	Severa/72-92	Bloqueio/110-130
Taxa de transferência de dados do fluxo malicioso em Mbps	9,87	3,78	0,56	0,00
Taxa de transferência de dados do fluxo legítimo em Mbps	6,47	8,99	14,72	15,04

Considerando os resultados, constata-se que após a geração do alerta pelo IDS, são necessários em média 7 segundos para que o controlador Of-IDPS perceba a ameaça, envie comandos ao *switch* e as ações programadas sejam aplicadas para cada alerta. Tem-se uma média de 3,33 segundos para que os fluxos estabilizem após a aplicação das restrições. Assim, decorrem em média 10 segundos da emissão do alerta até a estabilização do tráfego. Alterando parâmetros do Of-IDPS, como por exemplo, o período de verificação de novos alertas gerados pelos IDS - configurado para 5 segundos nos experimentos - pode-se diminuir esse tempo. Por fim, a reação automática em 7 segundos ao problema de segurança e efetivamente em 10 segundos é considerada positiva se comparado ao tempo de reação que um administrador humano levaria para identificar e corrigir o problema.

5.1.1.2 Experimentos 2 - Reação contra ataques de negação de serviço

Ataques de negação de serviço são uns dos principais problemas quando o assunto é segurança de redes de computadores, pois esses normalmente causam grandes distúrbios na rede e são complexos de serem resolvidos (ver Seção 2.1.1.2).

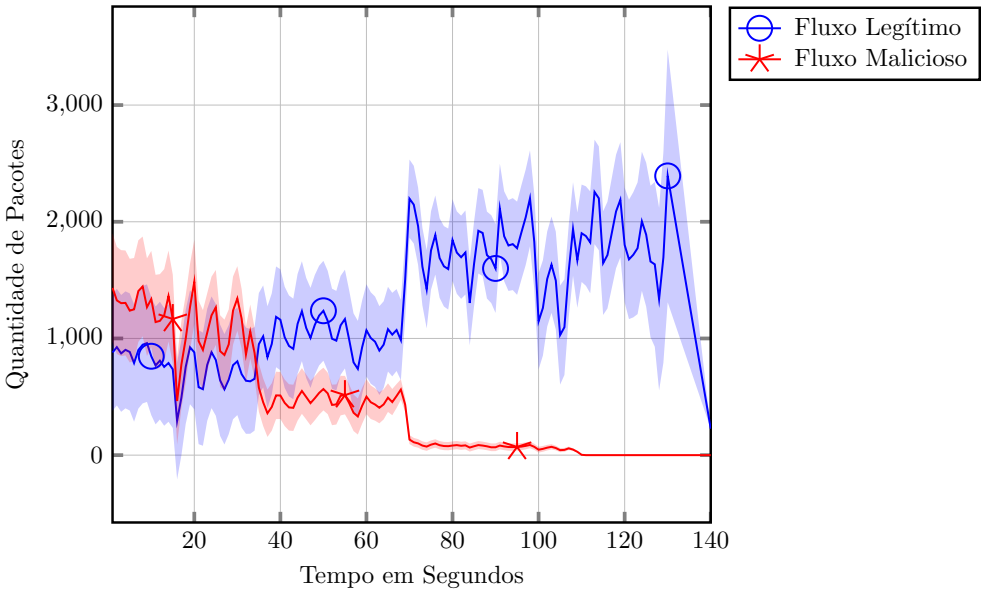


Figura 5.3: *Experimento 1 - Média e desvio padrão dos fluxos legítimo e malicioso.*

No geral, as redes podem tanto ser alvo de ataques DoS quanto utilizadas como fonte/origem desse tipo de ataque, pois não é incomum que computadores de LAN sejam infectados por vírus que tornam os *hosts* da rede "zumbis", prontos para executarem ataques DoS ao comando de *crackers*, o que pode influenciar negativamente todas as partes envolvidas no ataque, incluindo o *host* e a rede utilizada como fonte do ataque.

Desta forma, esta seção apresenta experimentos que submetem a arquitetura autonômica proposta a uma série de ataques DoS e DDoS. O objetivo desses experimentos é:

1. Avaliar se a metodologia proposta consegue analisar os alertas de segurança obtidos durante os ataques e gerar autonomicamente regras de segurança para mitigar ataques DoS e DDoS;

2. Observar se o Of-IDPS apresenta um comportamento uniforme na mitigação de ataques, independente do número de pacotes maliciosos tratados;

3. Analisar qual é um bom tempo de retenção dos alertas de segurança para a memória curta do Of-IDPS;

4. Observar se as regras de segurança criadas interferem em fluxos de redes legítimos;

A seguir são apresentados os experimentos com ataques DoS e DDoS.

5.1.1.2.1 Experimento 2.1 - DDoS variando o número de pacotes maliciosos e o tempo de retenção da memória curta

Este experimento consiste em realizar ataques DDoS de fonte desconhecida (com IPs falsos - *spoofing*) no *host* 1 do cenário de rede da Figura 5.1.

Dos objetivos citados anteriormente para os experimentos com ataques DoS, o presente experimento contempla três desses: (1) avaliar a efetividade do Of-IDPS em combater o ataque; (2) observar o comportamento do Of-IDPS independente do número de pacotes maliciosos e (3) analisar um bom tempo de retenção de alertas para a memória curta.

O objetivo dos experimentos que variam a quantidade de pacotes enviados durante o ataque é observar se o Of-IDPS apresenta um comportamento uniforme, na mitigação de ataques, independente do número de pacotes maliciosos tratados. Já os experimentos alterando o tempo de lembrança da memória curta, têm por objetivo analisar se a variação desse tempo pode produzir comportamentos

discrepantes que causem imprevistos na mitigação de problemas, bem como descobrir qual tempo é mais adequado para a criação de regras autonômicas que ajudem a sanar problemas em redes locais.

Para os experimentos variando a quantidade de pacotes enviados do atacante para a vítima, foram realizados testes com: 5.000, 10.000 e 20.000 pacotes, em cenários de rede com e sem o Of-IDPS. Ainda, para cada um dos experimentos que altera a quantidade de pacotes na rede com Of-IDPS, também foram realizados experimentos adicionais com a variação de tempo de retenção dos alertas na memória curta de 10, 30 e 60 segundos. Ou seja, o Experimento 2.1 é composto de 12 sub-experimentos, que são representados pela primeira e segunda coluna da Tabela 5.2. Assim primeiramente, são realizados testes enviando-se 5.000 pacotes maliciosos (ver primeira coluna da Tabela 5.2) em uma rede que emprega o Of-IDPS com uma configuração que retém os alertas de segurança recém gerados por 10 segundos na memória curta. Na sequencia, o mesmo teste com 5.000 pacotes é realizado nos cenários de rede com a memória curta configurada para 30 e 60 segundos, bem como, em um cenário de rede que não utiliza o Of-IDPS e que serve de experimento de controle (ver segunda coluna da Tabela 5.2). Depois a mesma sequência de configuração de cenários de rede é repetida para ataques com 10.000 e 20.000 pacotes maliciosos.

Os ataques são gerados utilizado-se a ferramenta Hyenae (HYENAE, 2016; SANS, 2016), que permite executar vários tipos de ataques DoS. Dentro do cenário de rede apresentado na Figura 5.1, cada experimento é repetido 30 vezes.

Tabela 5.2: *Experimento 2.1 - Número de pacotes enviados/recebidos por atacante e vítima durante ataque DDoS.*

Número de pacotes maliciosos enviados	Tempo da memória curta (em segundos)	Número de pacotes tratados durante ataques DDoS		Δ_{VA} (%)	Δ_{VV} (%)
		Atacante	Vítima		
5.000	10	5.017	1.258	74,9	87,4
	30	5.017	957	80,9	90,4
	60	5.035	1.044	79,3	89,5
	Experimento de controle*	5.020	9.965	-98,5	0,0
10.000	10	10.147	1.135	88,8	94,3
	30	10.200	1.008	90,1	94,9
	60	10.265	1.216	88,2	93,9
	Experimento de controle*	10.019	19.908	-98,7	0,0
20.000	10	20.173	991	95,1	97,5
	30	20.323	1.080	94,7	97,3
	60	20.195	991	95,1	97,5
	Experimento de controle*	20.069	39.886	-98,7	0,0

Δ_{VA} - Redução percentual dos pacotes maliciosos evitados pela vítima em relação ao atacante.
Δ_{VV} - Redução percentual de pacotes maliciosos tratados pela vítima com Of-IDPS em relação a vítima do experimento de controle, para cada experimento variando o número de pacotes maliciosos.
* - Executado em uma rede sem Of-IDPS (sem proteção), assim não há memória curta.

A Tabela 5.2 também apresenta os dados gerados por cada um dos sub-experimentos citados anteriormente. Sendo que a terceira e quarta coluna apresentam a quantidade de pacotes tratados respectivamente por atacante e vítima em cada um dos sub-experimentos. A penúltima coluna (Δ_{VA}) apresenta o percentual de pacotes evitados pelas vítimas em relação aos seus respectivos atacantes. A última coluna (Δ_{VV}) apresenta a redução em termos percentuais dos pacotes tratados pelas vítimas protegidas pelo Of-IDPS em relação a respectiva vítima[1] do experimento de controle, para cada experimento que varia a quantidade de pacotes.

Analisando os dados da Tabela 5.2 é possível observar que a arquitetura proposta consegue mitigar os ataques em todos os sub-experimentos e isso é melhor observado na coluna Δ_{VA}, que representa a redução em termos percentuais dos pacotes enviados/recebidos pela vítima em relação ao atacante. Por exemplo, no experimento com 10.000 pacotes utilizando o Of-IDPS com memória curta de 60 segundos, o atacante gera 10.265 pacotes, mas a vítima trata apenas 1.216 desses pacotes, o que representa uma diminuição de 88,2% do lado da vítima, o que pode ser considerado um caso médio, já que a média entre todos os experimentos usando o Of-IDPS foi de 87,3%. Mas, no melhor caso a vítima conseguiu evitar 95,1% dos pacotes maliciosos enviados pelo atacante, o que ocorreu

[1] Vítima da rede de controle que enviou a mesma quantidade de pacotes da vítima que está sendo analisada.

tanto usando a memória de 10 quanto a de 60 segundos nos experimentos com 20.000 pacotes. Na coluna Δ_{VA} os experimentos de controle apresentam resultados negativos devido ao fato da vítima tratar mais pacotes que o atacante, isso ocorre pois o acatante usa IPs falsos para executar o ataque, assim o pacote de resposta da vítima não retorna para o atacante. Contudo, resultados ainda mais expressivos são apresentados na coluna Δ_{VV}, que compara as vítimas que usam o Of-IDPS com as vítimas que não utilizam (experimento de controles), nessa comparação a redução média do número de pacotes maliciosos foi de 93,7%, sendo o pior caso 87,4% no experimento com 5.000 pacotes e memória curta com 10 segundos e o melhor caso 97,5% nos experimentos com 20.000 pacotes com e memória curta com 10 e 60 segundos.

Através dos sub-experimentos apresentados aqui, também constata-se que a variação do número de pacotes enviados não influencia drasticamente o comportamento da arquitetura. Por exemplo, os gráficos da Figura 5.4 apresentam o fluxo médio de pacotes enviados/recebidos pelo atacante e vítima nos experimentos com memória curta de 30 segundos. Analisando os dados do atacante (linha laranja com quadrado) e vítima (linha vermelha com círculo) da rede sem Of-IDPS, verifica-se a contundência dos ataques DoS, pois a rede é inundada pelos pacotes maliciosos e, por consequência, os recursos da rede e da vítima são comprometidos. Já examinando os dados do acatante (linha azul com asterisco) e vítima (linha verde com triângulo) que utilizam o Of-IDPS, é possível comprovar que a criação de regras autonômicas proposta consegue mitigar os ataques DoS, pois com cerca de 5 segundos de seu início as medidas reativas começam a surtir efeito, restringindo a taxa de transferência de dados dos pacotes maliciosos, o que notoriamente ajuda a conservar recursos da rede e proteger a vítima.

Complementarmente aos sub-experimentos anteriores e ainda no que se refere a análise da quantidade de pacotes, foi realizado um experimento adicional no qual foram enviados 1 (um) milhão de pacotes durante o ataque DDoS, isso foi feito em uma rede comum (sem Of-IDPS) e em uma rede com Of-IDPS configurada com a memória curta de 30 segundos. A Figura 5.5 apresenta a média e o desvio padrão dos fluxos de rede gerados durante esse experimento, na qual nota-se que o comportamento dos fluxos obtidos são similares aos respectivos fluxos apresentados nos sub-experimentos anteriores (ver Figura 5.4). Por exemplo, observa-se que a vítima (linha verde com triângulo) apresenta apenas um pequeno pico[2] de pacotes no inicio do ataque, depois disso os pacotes maliciosos destinados à vítima são mitigados pelo Of-IDPS, tal como ocorreu anteriormente nos sub-experimentos com 5.000, 10.000 e 20.000 pacotes (Figura 5.4). No que se refere a quantidade média de pacotes enviados/recebidos em cada fluxo desse experimento, o atacante da rede sem Of-IDPS tratou 1.002.097 pacotes e o acatante da rede com Of-IDPS tratou 1.007.333. Já a vítima da rede sem Of-IDPS enviou/recebeu 1.992.490 pacotes e a vítima da rede protegida pelo Of-IDPS tratou apenas 1.035. Ou seja, a vítima da rede que utilizou Of-IDPS apresentou uma redução de 99,948% na quantidade de pacotes maliciosos enviados/recebidos em relação a vítima da rede de controle.

Quanto à análise do tempo empregado na memória curta, constatou-se que nos experimentos com 5.000 e 10.000 pacotes, os melhores resultados foram com a retenção de alertas recentes por 30 segundos, pois nesses foram evitados respectivamente 80,9% e 90,1% dos pacotes maliciosos enviados pelos atacantes. No experimento com 20.000 pacotes, a memória de 30 segundos ficou apenas 0,4 pontos percentuais atrás do melhor caso que ocorreu tanto com as memórias de 10 segundos quanto com a de 60 segundos (ver Tabela 5.2).

Portanto, dê acordo com os objetivos apontados para este experimento, no inicio dessa subseção, concluí-se que:

1. A criação de regras autonômicas usando os métodos propostos consegue mitigar ataques DDoS;

2. A variação no número de pacotes maliciosos tratados pelo Of-IDPS não gerou nenhuma variação significativa nos resultados obtidos nos sub-experimentos. Inclusive, analisando os sub-experimentos da Figura 5.4 e do experimento com 1 milhão de pacotes (Figura 5.5), observa-se

[2] O pico da vítima com Of-IDPS é menor na Figura 5.5, pois sua escala está em minutos e não em segundos.

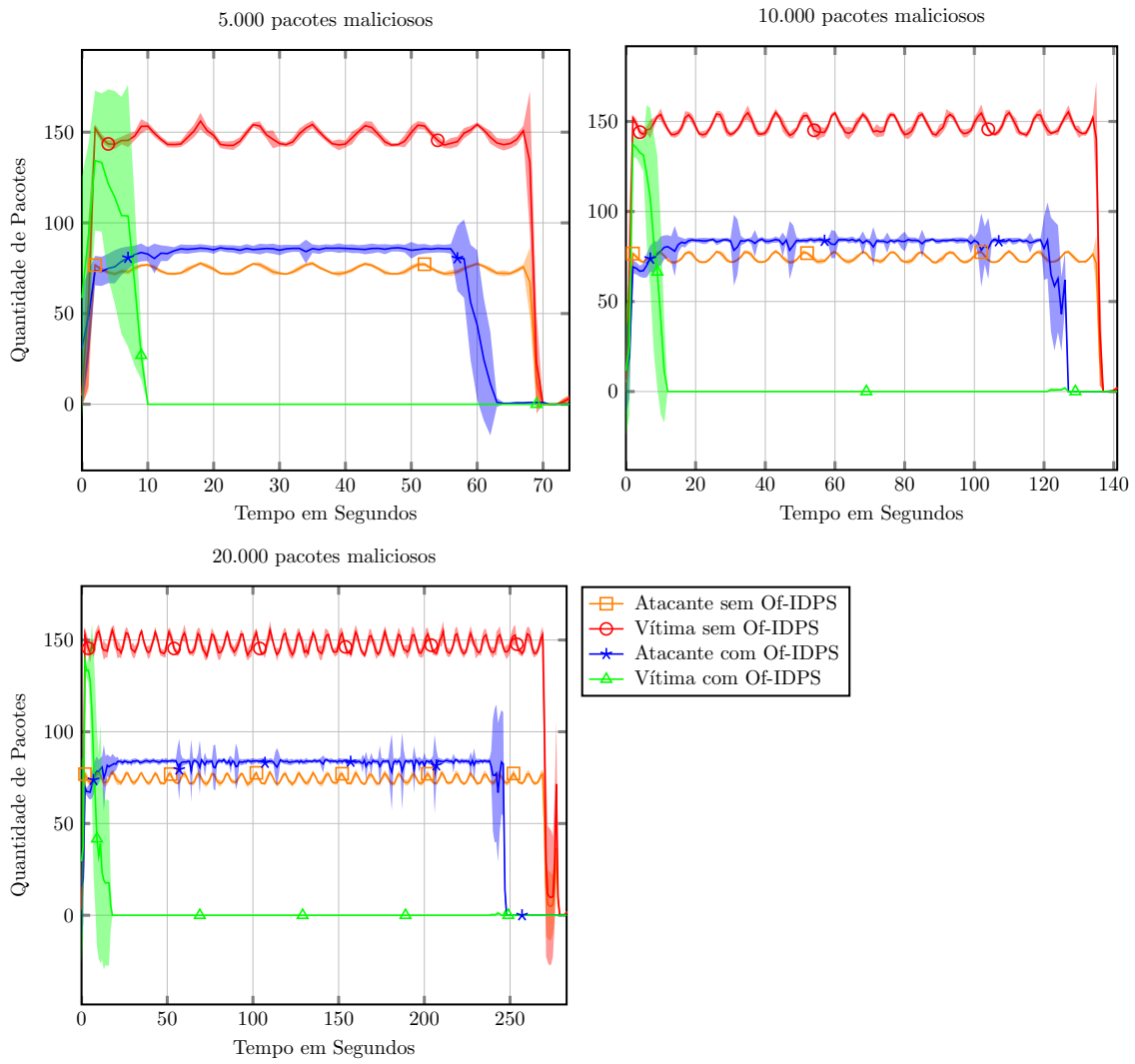


Figura 5.4: *Experimento 2.1 - Média e desvio padrão dos sub-experimentos variando a quantidade de pacotes do ataque DDoS e utilizando memória curta de 30 segundos.*

que redes utilizando o Of-IDPS possuem tendência de obter resultados ligeiramente melhores ao se aumentar o número de pacotes em ataques DoS, pois os fluxos de rede referentes ao ataque ficam com sua taxa de transferência de dados reduzida por mais tempo;

3. A mudança no tempo de retenção dos alertas da memória curta também não apresentou variações muito contundentes nos resultados. Todavia conclui-se que o uso de 30 segundos para a memória curta é uma boa opção, já que essa configuração apresentou resultados melhores nos testes com 5.000 e 10.000 pacotes maliciosos e ficou apenas 0,4 pontos percentuais no teste com 20.000 pacotes. Além do que a memória curta deve lidar com ataques recentes e portanto é natural que essa memória não retenha alertas passados por um longo período de tempo, para evitar o problema de *overfitting*(Harrington, 2012). Devido isso, o restante dos experimentos com o Of-IDPS realizados neste trabalho, serão executados utilizando uma configuração de tempo da memória curta com 30 segundos.

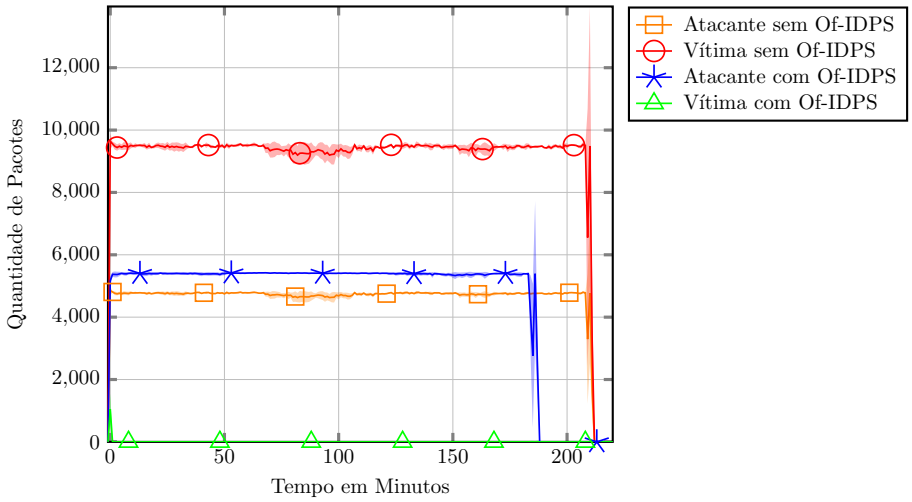


Figura 5.5: *Experimento 2.1 - Média e desvio padrão do experimento com 1 milhão de pacotes em um ataque DDoS.*

5.1.1.2.2 Considerações em relação aos experimentos com fluxos de redes maliciosos e legítimos

Os sub-experimentos do Experimento 2.1 demostraram que o Of-IDPS e a metodologia proposta consegue efetivamente conter os fluxos maliciosos gerados por ataques DDoS, independente do número de pacotes maliciosos enviados. Bem como, definiu um bom tempo para reter alertas de segurança recém identificados na memória curta. Já os experimentos com DoS apresentados nas próximas subseções, têm como objetivos (1) continuar avaliando a efetividade do Of-IDPS frente aos problemas com DoS, mas principalmente, (2) observando se as medidas reativas, geradas pela metodologia, não influenciam na transmissão de fluxos de redes considerados legítimos (não relacionados com problemas de segurança).

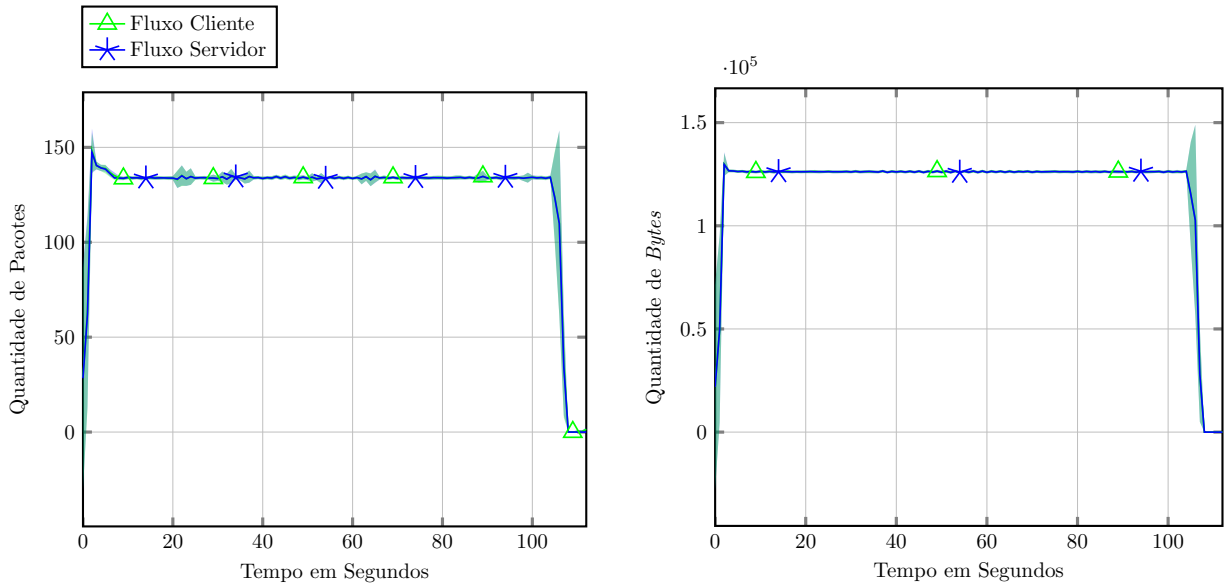
Para esses experimentos serão utilizadas as ferramentas Hping3 (Hping3, 2016), para gerar ataques DoS e o Wget (Wget, 2016) acessando um servidor HTTP Apache (Apache, 2016a), para representar os fluxos legítimos da rede. O Hping3 foi escolhido para avaliar o Of-IDPS frente a outras ferramentas utilizadas para ataques DoS, que não só o Hyenae. Quanto aos testes dos fluxos legítimos, escolheu-se o servidor HTTP Apache, pois esse é utilizado comumente na Internet. Já o Wget é um software popular dentre administradores de sistema UNIX, que permite recuperar arquivos (*download*) de servidores HTTP, HTTPS (*HyperText Transfer Protocol Secure*) e FTP.

Em todos os experimentos com fluxos legítimos, o Wget tentará recuperar do servidor HTTP um arquivo de 12 MB (*Megabyte*) e simultaneamente o comando Hping3 irá realizar um ataque DoS

de inundação enviando 20.000 pacotes TCP maliciosos, com uma área de dados de 1.000 *bytes* cada. O objetivo do ataque é ocupar toda a largura de banda da rede e causar distúrbios em outros fluxos de rede, no caso nos fluxos legítimos. Para facilitar o armazenamento, processamento, compilação e análise dos experimentos, a rede utilizada foi limitada a uma taxa de transferência de dados de 1Mbps - caso contrário faz-se necessário muito espaço em disco rígido para armazenar todos os testes e também é requerido muito tempo de processamento para analisar os dados gerados.

Para melhor observação do comportamento dos fluxos legítimos e maliciosos, as Figuras 5.6, 5.7 e 5.8 apresentam a média e o desvio padrão dos fluxos em experimentos de controle, nos quais os fluxos: (1) cliente/servidor (Wget/Apache), (2) atacante/vítima DoS e (3) atacante/vítima DDoS são executados isoladamente. Ou seja, o objetivo desses experimentos de controle é apresentar o comportamento desses fluxos sem interferência de outros fluxos. A Tabela 5.3 apresenta a quantidade total de pacotes e *bytes* transmitidos em cada experimento, bem como a taxa média em Bps (*Bytes* por segundo) e pps (pacotes por segundo)[3] de cada fluxo.

Assim, a Figura 5.6 representa os fluxos gerados pelo Wget/Apache e que são considerados legítimos, sendo que o gráfico 5.6-a apresenta a média e o desvio padrão quantificados em pacotes de rede e o gráfico 5.6-b o mesmo em *bytes*. Então, observa-se na Figura 5.6 que assim que o cliente (linha verde com triangulo) inicia a transferência de dados, o servidor (linha azul com asterisco) responde prontamente e os dois fluxos permanecem a pico durante toda a comunicação. A Tabela 5.3 mostra que tanto cliente quanto servidor enviaram aproximadamente 13 MB, em 14.000 pacotes a uma taxa de 117 Bps e 126 pps.



(a) Em Pacotes. (b) Em *Bytes*.

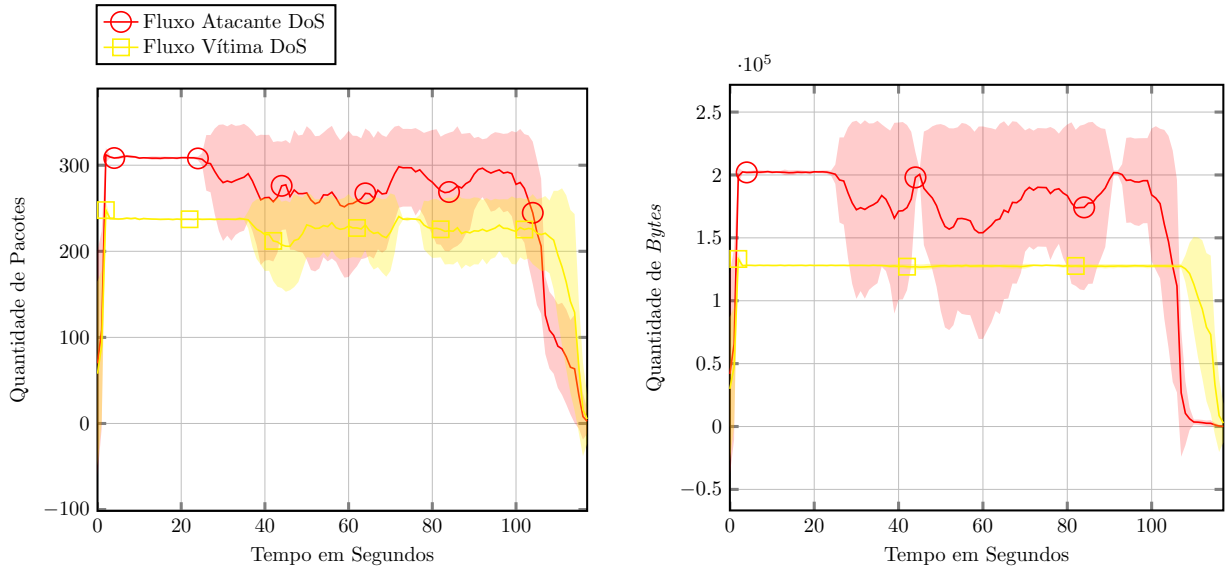
Figura 5.6: *Média e desvio padrão do fluxo legítimo em (a) pacotes e (b) bytes*

A Figura 5.7 representa os fluxos maliciosos gerados pelo atacante (linha vermelha com circulo) e vítima (linha amarela com quadrado), durante os ataques DoS. Nesse cenário, também é possível notar que assim que o atacante inicia o envio de pacotes, a vítima responde prontamente e continua assim durante todo o ataque. Nesses experimentos, o atacante é configurado para enviar 20.000 pacotes maliciosos para a vítima, que naturalmente responderá ao pacote TCP malicioso, o que potencializa os efeitos do ataque. Em outras palavras, nesse ataque DoS a vítima recebe 20.000 pacotes e responde, com pelo menos, mais 20.000 pacotes para o atacante, totalizando 40.000 pacotes, tanto do lado da vítima quanto do lado do atacante. Contudo, como é um ataque de saturação de *link* muitos pacotes serão perdidos/descartados. Assim, é possível verificar na Tabela 5.3 que o

[3]Devido ao fato do ataque ser medido pela quantide de pacotes enviados do atacante para a vítima, será utilizado a medida de pacotes por segundo.

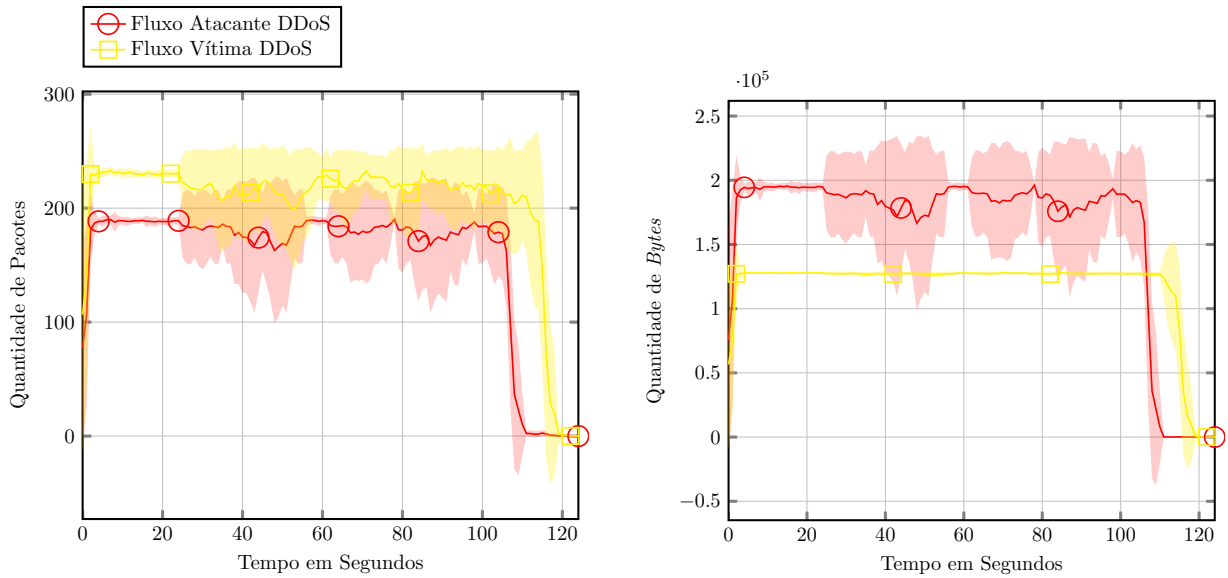
Tabela 5.3: *Dados dos experimentos com fluxo legítimo e fluxos maliciosos (DoS/DDoS) executados isoladamente.*

Fluxo	Quantidade Total		Bps	Pps
	Bytes	Pacotes		
Cliente	13.321.041	14.188	117.885	126
Servidor	13.321.544	14.198	117.890	126
Atacante DoS	19.381.814	30.588	164.253	259
Vítima DoS	14.375.949	25.756	121.830	218
Atacante DDoS	19.995.325	19.450	161.253	156
Vítima DDoS	14.629.295	25.351	117.034	203



(a) Em Pacotes. (b) Em *Bytes*.

Figura 5.7: *Média e desvio padrão do fluxo DoS em (a) pacotes e (b) bytes*



(a) Em Pacotes. (b) Em *Bytes*.

Figura 5.8: *Média e desvio padrão do fluxo DDoS em (a) pacotes e (b) bytes*

ataque conseguiu saturar a rede a ponto de descartar até mesmo pacotes maliciosos, pois o atacante enviou/recebeu 30.588 pacotes, o que gerou 19 MB na rede a uma taxa de 164 Bps e 259 pps. A

vítima tratou (enviou/recebeu) 25.756 pacotes, 14 MB, a uma taxa de 121 Bps e 218 pps. Outra forma, de se constatar a sobrecarga da rede, ocasionada pelo ataque DoS, é observando os distúrbios gerados na média e desvio padrão dos fluxos atacante e vítima da Figura 5.7.

A Figura 5.8 apresenta os efeitos do ataque DDoS. Nesse ataque os pacotes de resposta da vítima não são retornados para o atacante, pois os pacotes maliciosos são gerados com endereços (IP de origem) falsificados e aleatórios. Esse comportamento do ataque DDoS pode ser constatado nos dados da Tabela 5.3, pois se comparado com o atacante do experimento com DoS, o atacante DDoS enviou/recebeu apenas 19.450 pacotes, 19 MB, a uma taxa de 161 Bps e 156 pps. Já a vítima tratou 25.351 pacotes, o que totalizou 14 MB, a uma taxa de 117 Bps e 203 pps. Ou seja, o atacante conseguiu sobrecarregar a rede enviando pouco menos de 20.000 pacotes com uma área de dados de 1.000 *bytes*. A vítima tentou receber e responder os pacotes do ataque, mas devido a saturação do *link*, essa conseguiu tratar apenas 25.351 pacotes.

A seguir são apresentados experimentos que inserem simultaneamente fluxos maliciosos e legítimos em cenários de rede que empregam o Of-IDPS, para observar a influência das contramedidas autonômicas frente a ambos fluxos.

Em todos os experimentos seguintes, que envolvem fluxos legítimos e ataques DoS e DDoS, o fluxo legítimo será interrompido abruptamente assim que o ataque terminar. Isso será feito, pois depois que o ataque terminar o fluxo legitimo retorna a sua normalidade, o que dificulta a análise quantitativa do fluxo legítimo, já que esse continuará normalmente sem interferências até o seu termino.

5.1.1.2.3 Experimento 2.2 - DoS em *hosts* e serviços de rede distintos

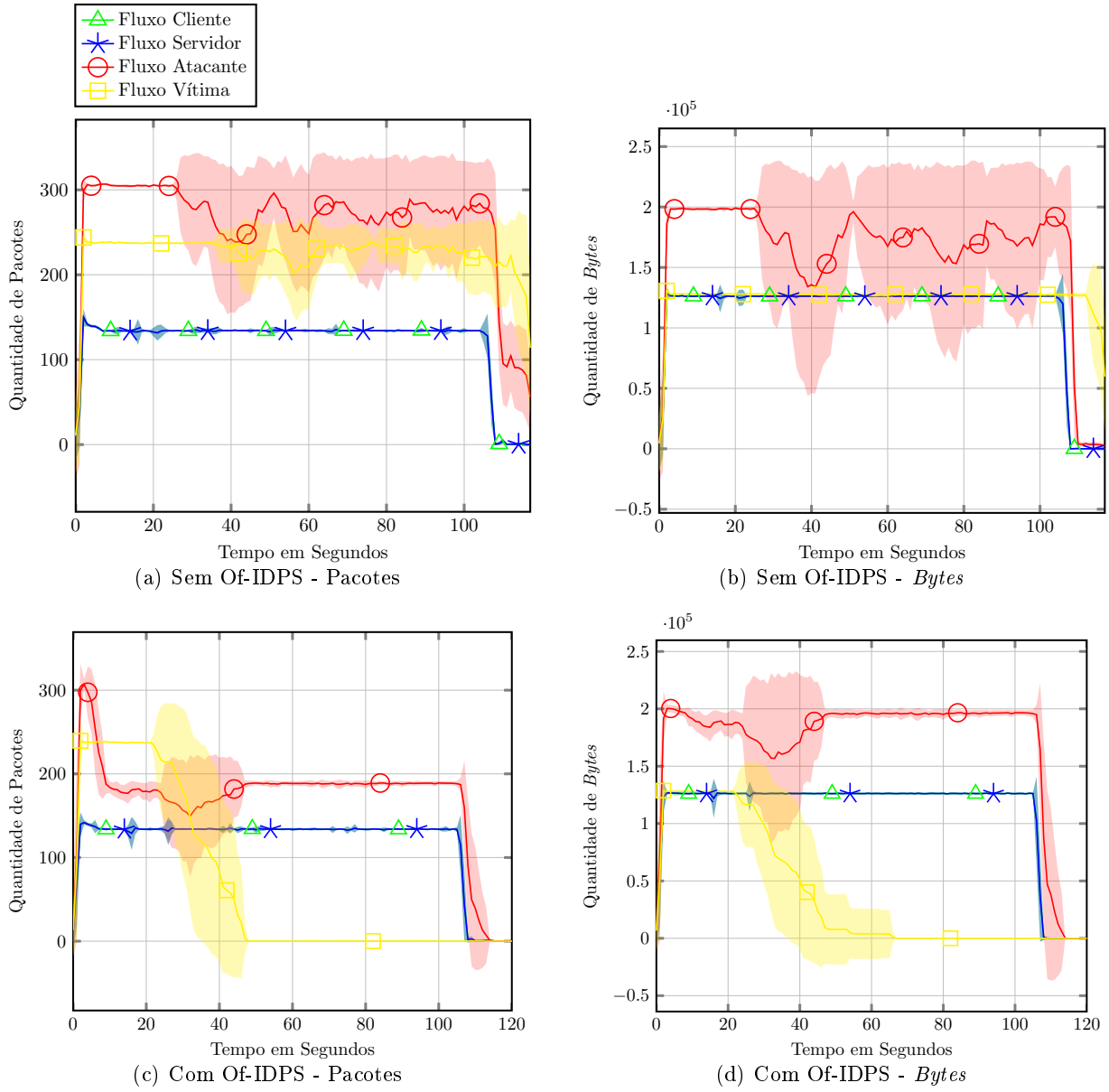
O objetivo deste experimento é verificar se as regras autonômicas criadas conseguem mitigar um ataque DoS de inundação de TCP, ocorrendo em um dado *host*/serviço sem causar distúrbios em outro *host*/serviço que não está sofrendo nenhum tipo de ataque.

Portando, para este experimento o fluxo legítimo é executado entre os *hosts* 3 (cliente) e 1 (servidor), da Figura 5.1. Já o fluxo malicioso é transmitido entre os *hosts* 4 (atacante) e 2 (vítima). O ataque DoS também é realizado em uma porta de rede diferente (TCP/99) da utilizada para a transmissão do fluxo legítimo (TCP/80), ou seja, tanto o *host* quanto o serviço de rede a ser atacado, são diferentes do *host* e serviço que transmitem fluxos de redes considerados legítimos.

Para uma melhor análise do experimento, a Figura 5.9 apresenta a média e o desvio padrão dos fluxos cliente (linha verde marcada com triângulos), servidor (linha azul com asteriscos), atacante (vermelha com círculos), vítima (laranja marcada com quadrados), sendo que os dois primeiro representam os fluxos legítimos e os dois últimos os fluxos maliciosos. A figura ainda está subdividida da seguinte forma: Média e desvio padrão da quantidade de pacotes transmitida por segundos na rede sem Of-IDPS (Figura 5.9-a) e com Of-IDPS (5.9-c); Média e desvio padrão da quantidade de *bytes* transmitidos por segundo na rede sem Of-IDPS (5.9-b) e com Of-IDPS (5.9-d). Os dados do experimento estão contabilizados na Tabela 5.4, que apresenta a quantidade média de pacotes e *bytes* enviados/recebidos, bem como, a média de *bytes* e pacotes por segundo em cada fluxo, tanto nos experimentos da rede sem, quanto na rede com Of-IDPS. Já a Tabela 5.5 é uma facilitadora quanto a análise de dados da Tabela 5.4, pois apresenta a diferença percentual dos dados/campos da rede com Of-IDPS em relação a rede sem Of-IDPS – *Esse esquema de tabelas, tal como, o mesmo padrão de subdivisão da figura, cores de linhas e marcações será mantida em todos os experimentos com DoS e DDoS.*

Analisando os fluxos do experimento sem Of-IDPS (Figura 5.9-a e 5.9-b), nota-se que assim que o atacante (Fluxo Atacante) inicia a ofensiva, a vítima (Fluxo Vítima) responde prontamente aos pacotes maliciosos comprometendo os recursos da vítima.

Contudo analisando os fluxos do experimento com Of-IDPS, que são representados pelas Figuras 5.9-c e 5.9-d, nota-se que o Of-IDPS detecta o problema e gera regras autonômicas contra o ataque em aproximadamente 4 segundos do inicio do ataque e com cerca de 8 segundos os pacotes maliciosos são mitigados, do lado do atacante. Isso pode ser percebido no pico existente no inicio do Fluxo



(a) Sem Of-IDPS - Pacotes

(b) Sem Of-IDPS - *Bytes*

(c) Com Of-IDPS - Pacotes

(d) Com Of-IDPS - *Bytes*

Figura 5.9: *Experimento 2.2 - Média e desvio padrão dos fluxos de rede, em pacotes e bytes*

Atacante da Figura 5.9-c, que indica que no inicio a vítima estava respondendo com pacotes TCP ao atacante, contudo com a instalação das regras autonômicas, a resposta da vítima é bloqueada, causando a redução no número de pacotes do fluxo do atacante.

A vítima da rede com Of-IDPS, só sente os efeitos da mitigação do ataque depois de aproximadamente 22 segundos, do inicio do ataque. Isso ocorre, pois mesmo depois do Of-IDPS mitigar o fluxo de ataque, o *switch* ainda continua entregando os pacotes maliciosos já recebidos (em memória) antes das regras de segurança terem sido instaladas.

Ainda quanto ao atacante da rede com Of-IDPS, esse continua a enviar pacotes maliciosos, todavia tais pacotes agora são barrados pelas regras de segurança instituídas pelo Of-IDPS, para conter o ataque. Assim, esses pacotes maliciosos não afetam mais negativamente a rede, pois não são encaminhados pelos *switches* da rede.

Utilizando os dados da Tabela 5.4 para comparar atacante e vítima da rede sem Of-IDPS, com os da rede utilizando Of-IDPS, constata-se a efetividade das regras autonômicas criadas.

O atacante na rede sem Of-IDPS enviou/recebeu 19 MB em 30.758 pacotes, com uma média de 162 Bps e 261 pps. Já o atacante da rede com Of-IDPS enviou/recebeu 20 MB em 20.259 pacotes,

Tabela 5.4: *Experimento 2.2 - Quantidade de pacotes/bytes e média de Bps/pps.*

Of-IDPS	Fluxo	Quantidade Total		Bps	Pps
		Bytes	Pacotes		
Sem	Cliente	13.321.956	14.210	112.898	120
	Servidor	13.322.395	14.218	112.902	120
	Atacante	19.149.512	30.758	162.284	261
	Vítima	14.710.680	26.473	124.667	224
Com	Cliente	13.321.880	14.186	110.098	122
	Servidor	13.322.326	14.194	110.102	117
	Atacante	20.377.806	20.259	168.412	167
	Vítima	4.635.024	8.236	38.306	71

Tabela 5.5: *Experimento 2.2 - Diferença em porcentagem entre experimentos sem e com Of-IDPS.*

Fluxo	Quantidade Total		Bps	Pps
	Bytes	Pacotes		
Cliente	■ 0,00	▼ 0,17	▼ 2,48	▲ 1,55
Servidor	■ 0,00	▼ 0,17	▼ 2,48	▼ 2,65
Atacante	▲ 6,41	▼ 34,13	▲ 3,78	▼ 35,77
Vítima	▼ 68,49	▼ 68,89	▼ 69,27	▼ 68,35

▲ Aumento percentual.
▼ Redução percentual.
■ Não apresentou diferença percentual.

com uma média de 168 Bps e 167 pps. Ou seja, o acatante da rede com Of-IDPS teve uma redução de 34,13% na quantidade de pacotes enviados/recebidos em relação ao atacante da rede de controle e teve uma redução de 35,77% na taxa de transmissão de pacotes por segundo. Isso ocorre, pois como já mencionado só houve um pico inicial do ataque e depois o fluxo foi bloqueado. Dessa maneira, os 20.259 pacotes registrados no atacante com Of-IDPS são basicamente os 20.000 pacotes programados para o ataque, adicionados à alguns pacotes de controle (ARP, TCP, etc) e a resposta da vítima, no inicio do ataque. Já os 30.758 pacotes tratados pelo atacante da rede de controle se referem aos 20.000 pacotes enviados pelo atacante, somados a resposta da vítima durante todo o decorrer do ataque e os pacotes de controle. Quanto ao aumento de 6,41% na quantidade de *bytes* no acatante com Of-IDPS, isso ocorre pois com as regras de segurança instaladas não há mais resposta da vitima para interferir no fluxo do atacante, ou seja, toda a largura de banda do *link* está dedicada ao atacante – é importante observar que esses pacotes maliciosos ficam restritos do *host* 4 ao *switch* OpenFlow (Figura 5.1) e não chegam na vítima (*host* 2).

A vítima na rede sem Of-IDPS envia/recebe 14 MB em 26.473 pacotes, a uma taxa média de 224 pps e 124 Bps. A vítima da rede com Of-IDPS, protegida pela metodologia proposta, trata 4 MB em 8.236 pacotes, apresentando uma taxa média de 71 pps e 38 Bps. Portanto, a vítima com Of-IDPS apresenta uma redução de 68% na quantidade de pacotes, *bytes* tratados, bem como na taxa de transferência de pacotes e *bytes* por segundo, em relação a vítima sem proteção. A taxa de 71 pps e 38 Bps, do lado da vítima na rede com Of-IDPS é obtida apenas do pico inicial do ataque, antes das regras autonômicas bloquearem os pacotes maliciosos.

Constatada novamente a efetividade do Of-IDPS em mitigar o ataque, agora será analisada se as reações autonômicas não geram perturbações que comprometam os fluxos legítimos. Assim, comparando o fluxo legítimo do cliente e servidor gerados na rede de controle (Figura 5.6), com o fluxo legítimo do cliente e servidor gerado no presente experimento nas redes sem e com Of-IDPS (Figura 5.9), observa-se que tanto o fluxo do cliente quanto do servidor possuem o mesmo padrão de comportamento em todos os cenários de testes (redes de controle, sem Of-IDPS e com Of-IDPS), o que indica que o fluxo legítimo na rede com Of-IDPS não foi afetado negativamente pela reação autonômica contra os fluxos maliciosos.

Analisando os dados gerados por cliente e servidor no experimento de controle (Tabela 5.3) e no cenário de ataque na rede com Of-IDPS (Tabela 5.4). Observa-se que no experimento de controle,

em média o cliente transmitiu 14.188 pacotes e o servidor 14.198, ambos ainda transmitiram 13 MB com uma taxa média de 117 Bps e 126 pps. Já na rede com Of-IDPS sob ataque, os fluxos legítimos atingiram uma média de 110 Bps e 122 pps no cliente, e 110 Bps e 117 pps no servidor, sendo transmitidos 13 MB em 14.186 pacotes pelo cliente e 13 MB em 14.194 pacotes pelo servidor. Então, na rede com Of-IDPS, mesmo sob ataque, não houveram diferenças significativas na média de *bytes*/pacotes transmitidos por cliente e servidor em relação a rede de controle, pois em ambos (cliente/servidor) a quantidade de *bytes* teve um aumento de apenas 0,01% e na quantidade de pacotes enviados/recebidos, houve uma redução de aproximadamente 0,02% no cliente e 0,03% no servidor. O que demonstra que as regras de segurança autonômicas criadas pelo Of-IDPS para mitigar o ataque não causaram distúrbios para os fluxos legítimos deste experimento.

Portanto, este experimento demostra que as regras de segurança criadas autonomicamente pelo Of-IDPS, para mitigar um ataque ocorrendo em um dado *host*/serviço, não afetam os fluxos legítimos de rede pertencentes a *hosts* e serviços de rede distintos ao ataque.

Neste experimento, devido ao fato dos fluxos maliciosos e legítimos serem encaminhados para *hosts* totalmente distintos e a rede utilizar *switches* para esses encaminhamentos (Tanenbaum e Wetherall , 2013), o ataque a principio não afeta diretamente o fluxo legítimo[4]. Por isso, o fluxo legítimo da rede sem Of-IDPS apresenta dados (Tabelas 5.4 e 5.5) bem similares ao da rede com Of-IDPS e do experimento de controle, o que também demonstra a conformidade dos experimentos com os efeitos do ataque em uma rede real.

A comparação direta entre o fluxo legítimo, do experimento de controle e da rede com e sem Of-IDPS, só é possível devido ao fato do fluxo legítimo terminar antes do fluxo malicioso, ou seja, não houve interrupção abrupta do fluxo legítimo devido ao termino do fluxo malicioso – como já mencionado anteriormente esse é um comportamento padrão adotado nesta pesquisa, para os experimentos envolvendo fluxos maliciosos e normais em ataques DoS e DDoS.

5.1.1.2.4 Experimento 2.3 - DoS: mesmo *host* com serviços de rede diferentes

O objetivo deste experimento é verificar se o Of-IDPS consegue mitigar um ataque DoS de inundação TCP ocorrendo em um dado serviço (porta[5]) de um *host* sem interferir negativamente em outros serviços de rede, em execução no mesmo *host*.

No presente experimento o servidor/vítima é o *host* 1 da Figura 5.1. Então, o fluxo legítimo do cliente será encaminhado do *host* 3 para a porta TCP/80 do *host* 1. Já o fluxo malicioso do atacante será enviado do *host* 4 para a porta TCP/99 do *host* 1. Ou seja, neste experimento serão destinados para uma mesma máquina (*host* 1) um fluxo legítimo (porta 80) e um fluxo malicioso (porta 99) e o Of-IDPS deve mitigar o ataque ocorrendo na porta 99 sem atrapalhar a transmissão da porta 80.

A Figura 5.10 apresenta a média e o desvio padrão dos fluxos de rede que compõem o presente experimento e a Tabela 5.6 a quantidade de *bytes* e pacotes enviados/recebidos, bem como a taxa média de *bytes* e pacotes por segundo de cada fluxo de rede do experimento. A Tabela 5.7, apresenta a diferença em porcentagem dos dados referentes aos experimentos com e sem Of-IDPS.

Analisando as Figuras 5.10-a e 5.10-b, que representam o ataque ocorrendo simultaneamente com a transmissão de um fluxo legítimo em uma rede sem proteção. Constata-se que o ataque DoS de inundação TCP consegue afetar o fluxo legítimo, mesmo esse estando em outra porta de rede, que não a porta que está sob ataque. Pois, assim que o ataque inicia a vítima é inundada por pacotes maliciosos (Fluxo Atacante e Fluxo Vítima), o que visivelmente reduz a taxa de transferência de dados dos fluxos legítimos (Fluxo Cliente e Fluxo Servidor das Figuras 5.10-a e 5.10-b).

Agora analisando o mesmo ataque, mas na rede com Of-IDPS (Figuras 5.10-c e 5.10-d), observa-se que no inicio do ataque a vítima responde prontamente aos pacotes maliciosos. Todavia, com aproximadamente 5 segundos do inicio do ataque, esse começa a ser contido e é mitigado com

[4]Como *switches* isolam a transmissão entre hosts distintos não há interferência direta, mas essa pode ocorrer indiretamente caso os recursos do *switch* (ex. processador e memória) sejam afetados pelo ataque.

[5]Camada de Aplicação (Tanenbaum e Wetherall, 2013).

aproximadamente 10 segundos do inicio do ataque. Quanto a vítima, essa começa a sentir os efeitos da mitigação do ataque com 20 segundos, mas os pacotes só cessam com aproximadamente 45 segundos (devido aos pacotes maliciosos que foram enviados pelo atacante antes da imposição das regras de segurança criadas pelo Of-IDPS e que foram armazenados no *switch*).

A eficácia da reação autonômica pode ser melhor observada comparando os dados do atacante e vítima da rede que utiliza Of-IDPS, com os dados da rede sem Of-IDPS, na Tabela 5.6. Sendo que o atacante na rede sem Of-IDPS envia/recebe 18 MB em 30.719 pacotes, a uma taxa de 152 Bps e 252 pps. A vítima desse mesmo cenário envia/recebe 14 MB em 26.513 pacotes, a uma taxa de 117 Bps e 217 pps.

No cenário de rede com Of-IDPS o atacante envia/recebe 20 MB em 20.473 pacotes a 173 Bps e 172 pps e a vítima, protegida pelo Of-IDPS, envia/recebe apenas 3 MB em 7.126 pacotes a uma taxa de 32 Bps e 60 pps. Desta forma, o atacante da rede com Of-IDPS teve uma redução de 10,80% e 33,35% (ver Tabela 5.7), na quantidade de *bytes* e pacotes enviados/recebidos, respectivamente. Quanto a taxa de *bytes* e pacotes por segundo, a redução foi respectivamente de 13,60% e 31,67%, em relação ao atacante da rede sem Of-IDPS. Mas principalmente o Of-IDPS conseguiu reduzir em pouco mais de 72% a taxa de *bytes* e pacotes por segundo, bem como a quantidade de *bytes* e pacotes maliciosos tratados pela vítima, quando comparada com a vítima da rede sem Of-IDPS.

Tabela 5.6: *Experimento 2.3 - Quantidade de pacotes/bytes e média de Bps/pps.*

Of-IDPS	Fluxo	Quantidade Total		Bps	Pps
		Bytes	Pacotes		
Sem	Cliente	1.951.173	2.416	15.993	20
	Servidor	1.940.020	2.312	15.902	19
	Atacante	18.634.441	30.719	152.741	252
	Vítima	14.319.759	26.513	117.375	217
Com	Cliente	10.594.146	10.438	89.026	88
	Servidor	10.592.468	10.404	89.012	87
	Atacante	20.647.723	20.473	173.510	172
	Vítima	3.880.375	7.126	32.608	60

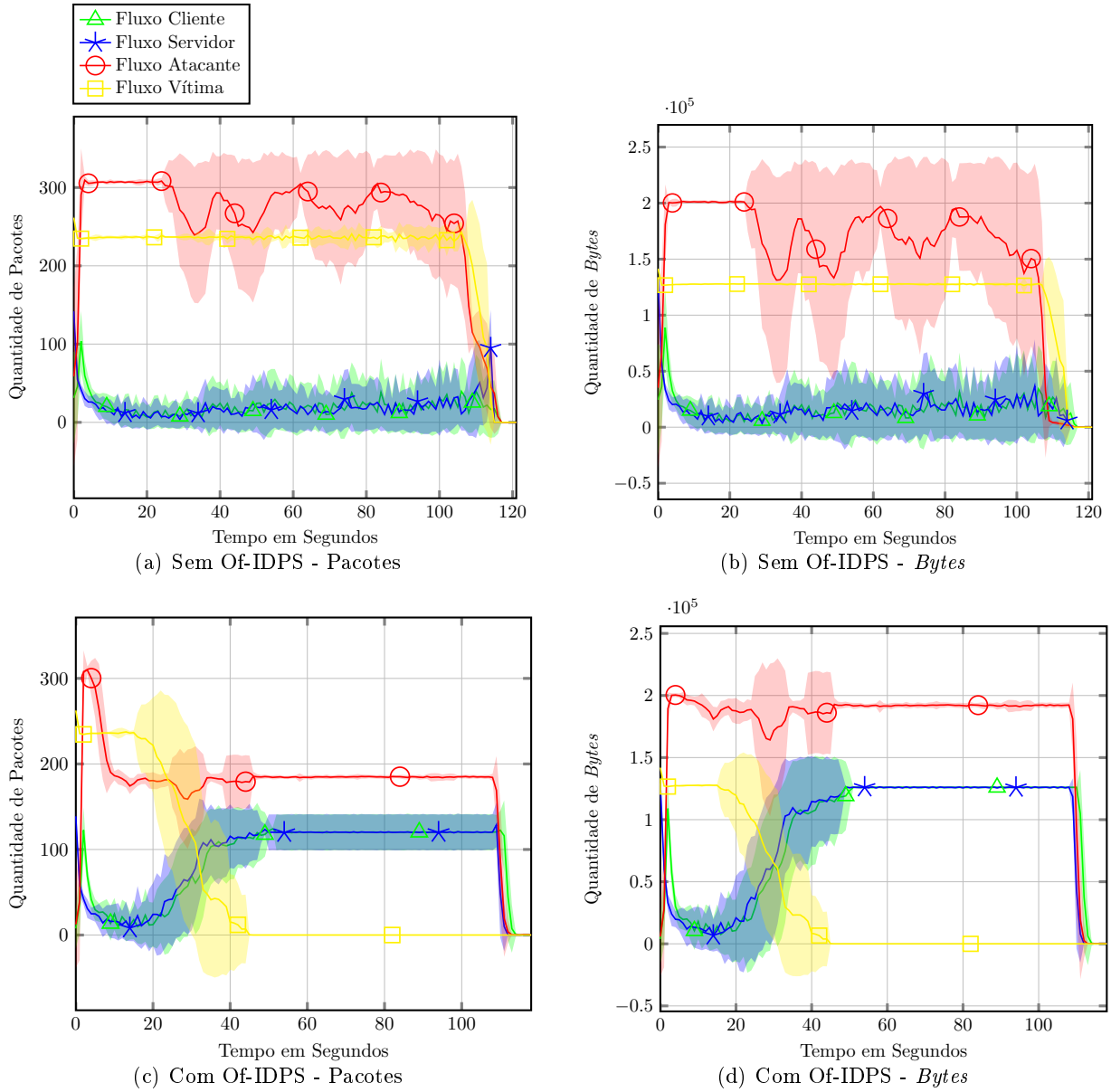
Tabela 5.7: *Experimento 2.3 - Diferença em porcentagem entre experimentos sem e com Of-IDPS.*

Fluxo	Quantidade Total		Bps (%)	Pps (%)
	Bytes (%)	Pacotes (%)		
Cliente	▲ 442,96	▲ 332,07	▲ 456,65	▲ 342,96
Servidor	▲ 446,00	▲ 350,02	▲ 459,76	▲ 361,36
Atacante	▲ 10,80	▼ 33,35	▲ 13,60	▼ 31,67
Vítima	▼ 72,90	▼ 73,12	▼ 72,22	▼ 72,44

▲ Aumento percentual.
▼ Redução percentual.

Quanto aos fluxos legítimos do experimento com Of-IDPS, observa-se nas Figuras 5.10-a e 5.10-b que os pacotes de dados, tanto do servidor quanto do cliente (Fluxo Servidor e Fluxo Cliente), são afetados inicialmente pelo ataque. Contudo, depois que as regras de segurança são estabelecidas pelo Of-IDPS e os pacotes maliciosos param de circular na rede, ambos fluxos (cliente/servidor) retomam o seu comportamento normal, ou seja, tornam-se similar ao fluxo de controle apresentado na Figura 5.6. Isso indica que o Of-IDPS consegue criar regras de segurança autonômicas para mitigar um ataque ocorrendo em um serviço de um dado *host*, sem interferir negativamente em outros serviços de rede em execução no mesmo *host*.

O comportamento do Of-IDPS em mitigar o ataque DoS e manter o bom funcionamento dos fluxos legítimos, pode ser melhor constatado através da análise dos dados da Tabela 5.6, pois no experimento sem Of-IDPS, cliente e servidor enviaram/receberam cada um aproximadamente 1,9 MB em pouco mais de 2.000 pacotes, a uma taxa média de 15 Bps e 20 pps, o que demonstra



(a) Sem Of-IDPS - Pacotes

(b) Sem Of-IDPS - *Bytes*

(c) Com Of-IDPS - Pacotes

(d) Com Of-IDPS - *Bytes*

Figura 5.10: *Experimento 2.3 - Média e desvio padrão dos fluxos de rede, em pacotes e bytes*

que o ataque DoS comprometeu o fluxo legítimo na rede sem Of-IDPS. Já o cliente e servidor do experimento com Of-IDPS, trataram separadamente 10,5 MB em pouco mais de 10.000 pacotes, em uma taxa de 89 Bps e 88 pps. O que de acordo com a Tabela 5.7, representa um aumento aproximado de 442% e 332%, respectivamente na quantidade de *bytes* e pacotes enviados e recebidos no cliente com Of-IDPS, em relação ao cliente sem. No servidor o aumento foi de 446% e 350%, respectivamente para a quantidade de *bytes* e pacotes tratados no servidor com Of-IDPS em relação a rede sem Of-IDPS. Já em relação a taxa de *bytes* e pacotes por segundo, essa apresentou um aumento de: 456% Bps e 342% pps no cliente e 459% Bps e 361% pps no servidor, em comparação da rede sem e com Of-IDPS. Ou seja, o Of-IDPS conseguiu conter os efeitos negativos do ataque tanto no fluxo cliente quanto no servidor.

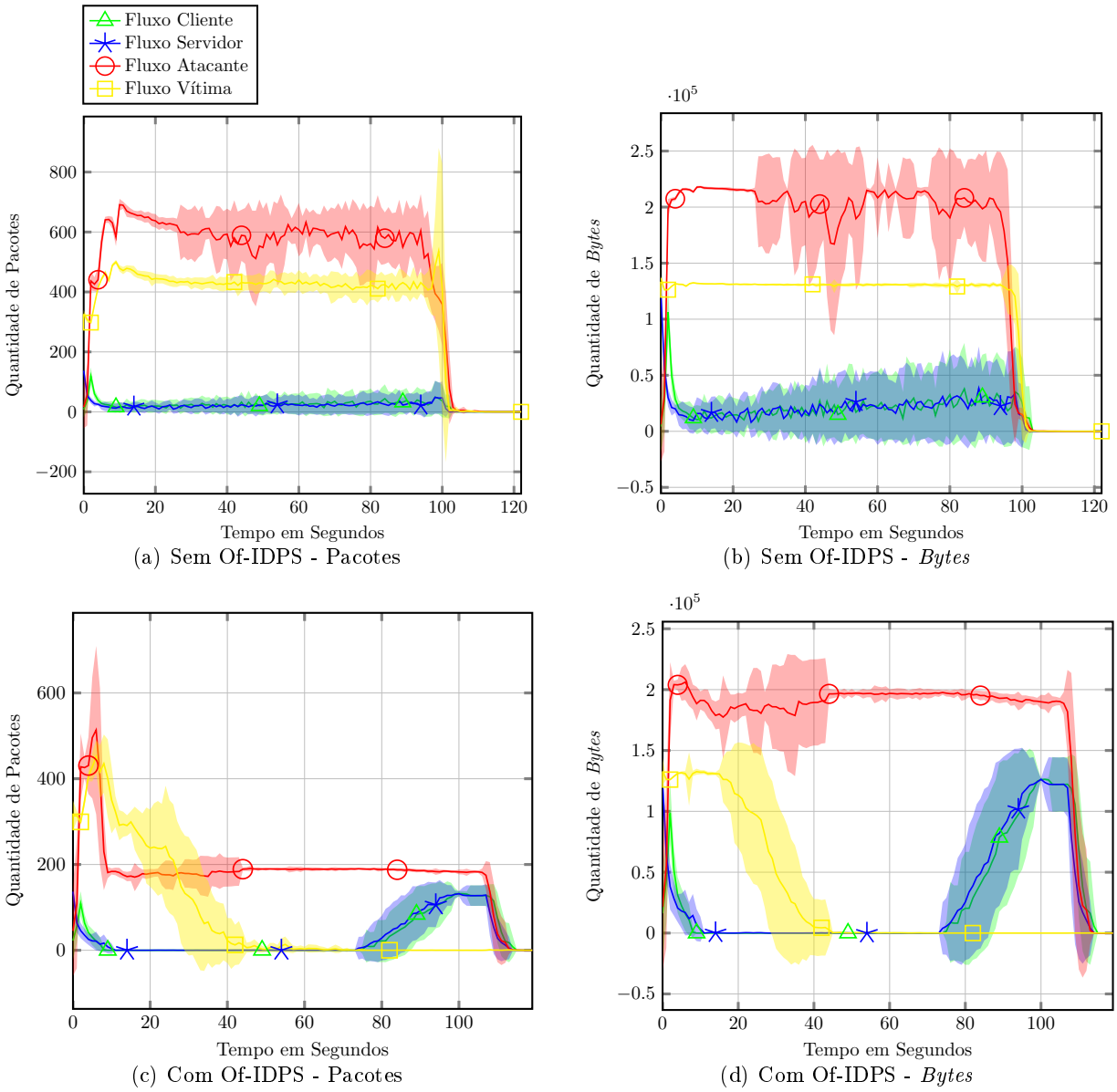
Portanto, este experimento demonstra que o Of-IDPS é capaz de mitigar o ataque DoS, que está ocorrendo em uma dada porta de um *host*, sem causar grandes distúrbios nos demais serviços de rede que estão em execução no mesmo *host* do ataque.

5.1.1.2.5 Experimento 2.4 - DoS: mesmo *host* e serviço de rede

O objetivo desse experimento é analisar o comportamento da reação autonômica do Of-IDPS quando há tanto fluxos legítimos quanto maliciosos sendo destinados simultaneamente a um mesmo serviço de rede de um dado *host*. Ou seja, a meta é verificar se as regras de segurança autonômicas conseguem mitigar, os fluxos maliciosos de um ataque DoS de inundação TCP, sem causar grandes distúrbios nos fluxos legítimos, mesmo com esses ocorrendo em um mesmo *host* e serviço de rede.

Neste experimento tanto o fluxo de rede malicioso quanto o legítimo são encaminhados para a porta TCP/80 do *host* 1, no qual está em execução um servidor HTTP. O atacante é o *host* 4, que faz uso do Hping3 para enviar pacotes maliciosos para a porta 80 do *host* 1. O *host* 3 é o cliente legítimo, que utiliza o Wget para acessar a porta 80 do *host* 1. O cenário de rede com os *hosts* utilizados para este experimento são ilustrados na Figura 5.1.

A Figura 5.11 apresenta a média e o desvio padrão dos fluxos gerados pelo presente experimento. A Tabela 5.8 mostra a quantidade e a taxa de transferência de dados dos fluxos cliente, servidor, atacante e vítima. Já a Tabela 5.9 exibe a relação em termos percentuais entre os dados da rede com e sem Of-IDPS.



(a) Sem Of-IDPS - Pacotes

(b) Sem Of-IDPS - *Bytes*

(c) Com Of-IDPS - Pacotes

(d) Com Of-IDPS - *Bytes*

Figura 5.11: *Experimento 2.4 - Média e desvio padrão dos fluxos de rede, em pacotes e bytes*

Analisando as Figuras 5.11-a e 5.11-b, constata-se que os efeitos do ataque na rede sem Of-IDPS, apresentados neste experimentos, são muito similares aos obtidos no experimento anterior (ver Figuras 5.10-a e 5.10-b).

Todavia, analisando os fluxos da rede com Of-IDPS (Figuras 5.11-c e 5.11-d), principalmente comparando com os fluxos legítimos de experimentos anteriores. Como por exemplo, o fluxo da Figura 5.6. Nota-se que a reação autonômica gerada para combater o ataque DoS causou distúrbios na transmissão de dados dos fluxos legítimos.

Então, examinando em detalhes os fluxos dos experimentos com Of-IDPS, observa-se que o atacante DoS (Fluxo Atacante na Figura 5.11-c) é contido em aproximadamente 8 segundos, depois de seu inicio. Ou seja, o Of-IDPS confeccionou autonomicamente regras de segurança para mitigar os pacotes maliciosos, de forma que esses não cheguem mais na porta TCP/80 do *host* 1, que é a vítima (Fluxo Vítima). Desta forma, a vítima para de receber os pacotes maliciosos depois de aproximadamente 44 segundos do inicio do ataque.

Contudo, em um primeiro momento a análise dos alertas de segurança a respeito do ataque DoS, feita pela memória curta do Of-IDPS, gera regras de segurança que bloqueiam todos os pacotes originados/destinados a porta TCP/80 do *host* 1, incluindo os fluxos legítimos.

A principio o bloqueio do fluxo legítimo pelo Of-IDPS, durou aproximadamente 64 segundos, ocorrendo entre 10 e 74 segundos do tempo decorrido nos testes (ver Fluxo Cliente e Servidor nas Figuras 5.11-c e 5.11-d). Todavia, em um segundo momento (após esses 64 segundos) o Of-IDPS consegue convergir as regras de segurança para mitigar apenas o fluxo malicioso originado pelo *host* 4 e destinado a porta TCP/80 do *host* 1, o que notoriamente faz com que os fluxos entre os *hosts* 1 e 3 sejam restabelecidos.

O tempo para o restabelecimento do fluxo cliente depois que o mesmo foi bloqueado e liberado, não depende exclusivamente do Of-IDPS. Isso, porque assim que o Wget percebe que a conexão foi interrompida, esse adota uma política de usar tempos aleatórios para verificar novamente a disponibilidade do servidor. Ou seja, o tempo de retomada do *download* do experimento depende também desse tempo do Wget. Em testes complementares utilizando o Iperf, para simular o fluxo legítimo, o tempo de retomada caiu para 45 segundos.

Tabela 5.8: *Experimento 2.4 - Quantidade de pacotes/bytes e média de Bps/pps.*

Of-IDPS	Fluxo	Quantidade Total		Bps	Pps
		Bytes	Pacotes		
Sem	Cliente	2.230.723	2.753	18.136	22
	Servidor	2.212.332	2.395	17.986	19
	Atacante	19.683.083	58.228	160.025	473
	Vítima	13.055.386	43.498	106.141	354
Com	Cliente	3.145.524	3.388	26.213	28
	Servidor	3.206.863	3.424	26.724	29
	Atacante	20.736.411	21.619	172.803	180
	Vítima	3.784.834	9.739	31.540	81

Tabela 5.9: *Experimento 2.4 - Diferença em porcentagem entre experimentos sem e com Of-IDPS.*

Fluxo	Quantidade Total		Bps	Pps
	Bytes	Pacotes		
Cliente	▲ 41,01	▲ 23,07	▲ 44,53	▲ 26,15
Servidor	▲ 44,95	▲ 42,95	▲ 48,58	▲ 46,53
Atacante	▲ 5,35	▼ 62,87	▲ 7,99	▼ 61,94
Vítima	▼ 71,01	▼ 77,61	▼ 70,28	▼ 77,05

▲ Aumento percentual.
▼ Redução percentual.

Quanto aos dados das Tabelas 5.8 e 5.9, nota-se que mesmo com o bloqueio dos fluxos legítimos, o Of-IDPS melhorou o desempenho do cliente/servidor sob ataque. Pois, o cliente teve um aumento

de 41% na quantidade de *bytes* enviados/recebidos e no servidor o aumento foi de 44%, em relação aos experimentos com a rede sem Of-IDPS. Ou seja, no experimento sem Of-IDPS o cliente tratou 2.2 MB a uma taxa de 18 Bps, já o cliente com Of-IDPS enviou 3.1 MB a 26 Bps. O servidor no experimento sem Of-IDPS enviou/recebeu 2.2 MB a 17 Bps e o servidor do experimento com Of-IDPS, sob ataque, tratou 3.2 MB a uma taxa de 26 Bps.

Ainda observando os dados das Tabelas 5.8 e 5.9, também nota-se que a vítima da rede com Of-IDPS teve uma redução de 71% na quantidade de *bytes* enviados/recebidos e 70% na média de *bytes* por segundo, se comparada com a vítima da sem Of-IDPS.

Desta forma, este experimento demonstra que a reação autonômica gerada para conter ataques DoS de inundação TCP pode causar distúrbios em fluxos legítimos, quando esses são destinados/originados para o mesmo *host* e serviço de rede que está sob ataque. Entretanto, conforme pode ser visto nos experimentos (Figuras 5.11-c e 5.11-d), esse distúrbio nos fluxos legítimos é momentâneo, ou seja, a metodologia empregada no Of-IDPS consegue refinar as regras de segurança a ponto de mitigar apenas os fluxos maliciosos, restabelecendo a transmissão de dados dos fluxos legítimos.

5.1.1.2.6 Experimento 2.5 - DDoS: mesmo *host* e serviço de rede

O objetivo deste experimento é praticamente o mesmo do Experimento 2.4, apresentado anteriormente. Ou seja, é analisar o comportamento da reação autonômica do Of-IDPS quando há tanto fluxos legítimos quanto maliciosos sendo destinados simultaneamente para um mesmo serviço e *host* da rede. Todavia, neste experimento a vítima é submetida a um ataque DDoS de inundação TCP e não a um ataque DoS.

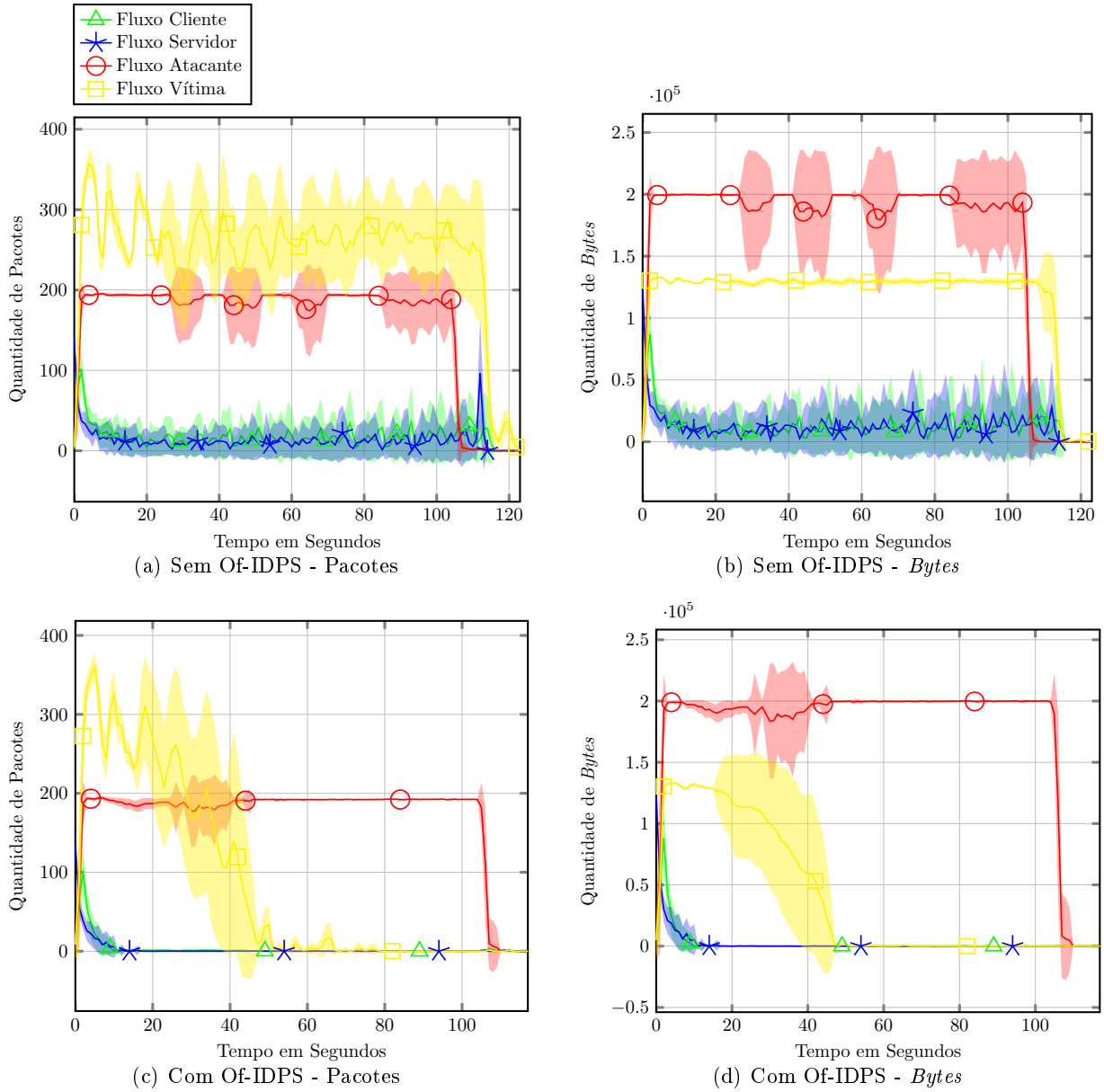
Este experimento é realizado no mesmo cenário de rede do Experimento 2.4, com a diferença que ao invés do atacante (*host* 4) utilizar um IP verdadeiro para realizar o ataque, esse utiliza diversos endereços falsos, para enganar a vítima (*spoofing*). Desta forma, do ponto de vista da vítima, ela está sendo atacada por diversos *hosts* (DDoS), o que aumenta consideravelmente a complexidade em se realizar a contenção efetiva do ataque.

A Figura 5.12 apresenta a média e o desvio padrão dos fluxos legítimos e maliciosos gerados pelo presente experimento. A Tabela 5.10 demonstra a taxa média de *bytes* e pacotes por segundo e a quantidade de *bytes* e pacotes enviados/recebidos por cada fluxo do experimento. Já a Tabela 5.11 mostra a diferença, em porcentagem, do experimento com Of-IDPS em relação ao experimento sem Of-IDPS.

Tabela 5.10: *Experimento 2.5 - Quantidade de pacotes/bytes e média de Bps/pps.*

Of-IDPS	Fluxo	Quantidade Total		Bps	Pps
		Bytes	Pacotes		
Sem	Cliente	1.552.369	2.153	12.519	18
	Servidor	1.531.660	1.663	12.352	15
	Atacante	20.316.751	19.784	163.845	165
	Vítima	14.563.366	30.332	117.447	245
Com	Cliente	255.997	377	2.169	3
	Servidor	307.253	355	2.604	3
	Atacante	20.689.050	19.966	175.331	169
	Vítima	4.611.798	10.050	39.083	85

Observando a Figura 5.12 constata-se que o Of-IDPS é capaz de conter o ataque DDoS, pois depois de aproximadamente 45 segundos de seu inicio, esse é totalmente mitigado. Contudo em virtude da complexidade do ataque, o Of-IDPS acaba restringindo completamente o acesso ao serviço de rede (porta TCP/80) da vítima sob ataque, o que inclui os fluxos legítimos. Isso ocorre porque em um ataque DDoS são empregados diversos IPs de origem. Então nesse caso, o Of-IDPS converge para regras de segurança que generalizam o *host* que está atacando o serviço de rede em questão. Desta forma, para conter o ataque e evitar o comprometimento dos recursos da vítima ou mesmo da rede, o Of-IDPS cria regras de segurança que mitigam todos os fluxos de rede relacionados



(a) Sem Of-IDPS - Pacotes

(b) Sem Of-IDPS - *Bytes*

(c) Com Of-IDPS - Pacotes

(d) Com Of-IDPS - *Bytes*

Figura 5.12: *Experimento 2.5 - Média e desvio padrão dos fluxos de rede, em pacotes e bytes*

com o ataque.

Comparando a rede sem Of-IDPS com a rede utilizando Of-IDPS, apresentados nas Tabelas 5.10 e 5.11, nota-se que tanto cliente quanto servidor da rede com Of-IDPS tiveram respectivamente 82% e 78% de seus pacotes restringidos pelas regras de segurança autonômicas que visavam conter o ataque.

No que se refere ao Fluxo Vítima, o Of-IDPS conseguiu mitigar 66% dos pacotes maliciosos enviados/recebidos em relação ao mesmo fluxo da rede de sem Of-IDPS.

Em relação a quantidade de pacotes enviados/recebidos pelos atacantes da rede sem Of-IDPS e da rede com Of-IDPS, essa foi praticamente a mesma em ambos experimentos, com uma diferença de 0,92% a mais na rede com Of-IDPS (ver Tabela 5.11).

Analisando ainda a Tabela 5.10, também nota-se que a quantidade de pacotes enviados/recebidos pelo atacante da rede sem Of-IDPS diminuiu em relação aos experimentos anteriores com DoS, como por exemplo no Experimento 2.4. Isso ocorre pelo fato já mencionado, de que os pacotes de resposta da vítima não retornam ao atacante, já que utilizam endereços IPs falsos.

Assim, este experimento demostra que o Of-IDPS consegue mitigar ataques DDoS, tal como já

Tabela 5.11: *Experimento 2.5 - Diferença em porcentagem entre experimentos sem e com Of-IDPS.*

Fluxo	Quantidade Total		Bps	Pps
	Bytes	Pacotes		
Cliente	▼ 83,51	▼ 82,48	▼ 82,67	▼ 82,63
Servidor	▼ 79,94	▼ 78,63	▼ 78,92	▼ 79,35
Atacante	▲ 1,83	▲ 0,92	▲ 7,01	▲ 2,63
Vítima	▼ 68,33	▼ 66,87	▼ 66,72	▼ 65,18

▲ Aumento percentual.
▼ Redução percentual.

havia sido demonstrado no Experimento 2.1 (Subseção 5.1.1.2.1). No entanto, devido a agressividade do ataque os fluxos relacionados com o *host*/serviço de rede sob ataque são afetados pelas regras de segurança geradas nas memórias sensorial e curta da metodologia proposta. Todavia, o uso da memória longa boa pode ajudar a evitar que fluxos conhecidos/legítimos sejam afetados nesses casos, tal como é demonstrado posteriormente no Experimento 2.7.

5.1.1.2.7 Experimento 2.6 - DDoS: mesmo *host* com serviços de rede diferentes

O objetivo deste experimento é verificar se o Of-IDPS é capaz de mitigar ataques DDoS de inundação de TCP ocorrendo em uma dada porta de rede, sem afetar negativamente outros serviços (portas) em execução no mesmo *host*, que está sendo atacado. Ou seja, este é similar ao Experimento 2.3 da Subseção 5.1.1.2.4, exceto pelo fato do presente ataque ser DDoS e não DoS.

Para este experimento é utilizado o mesmo cenário de rede do Experimento 2.3, mas com o atacante configurado para efetuar ataques DDoS, tal como no experimento anterior (Subseção 5.1.1.2.6).

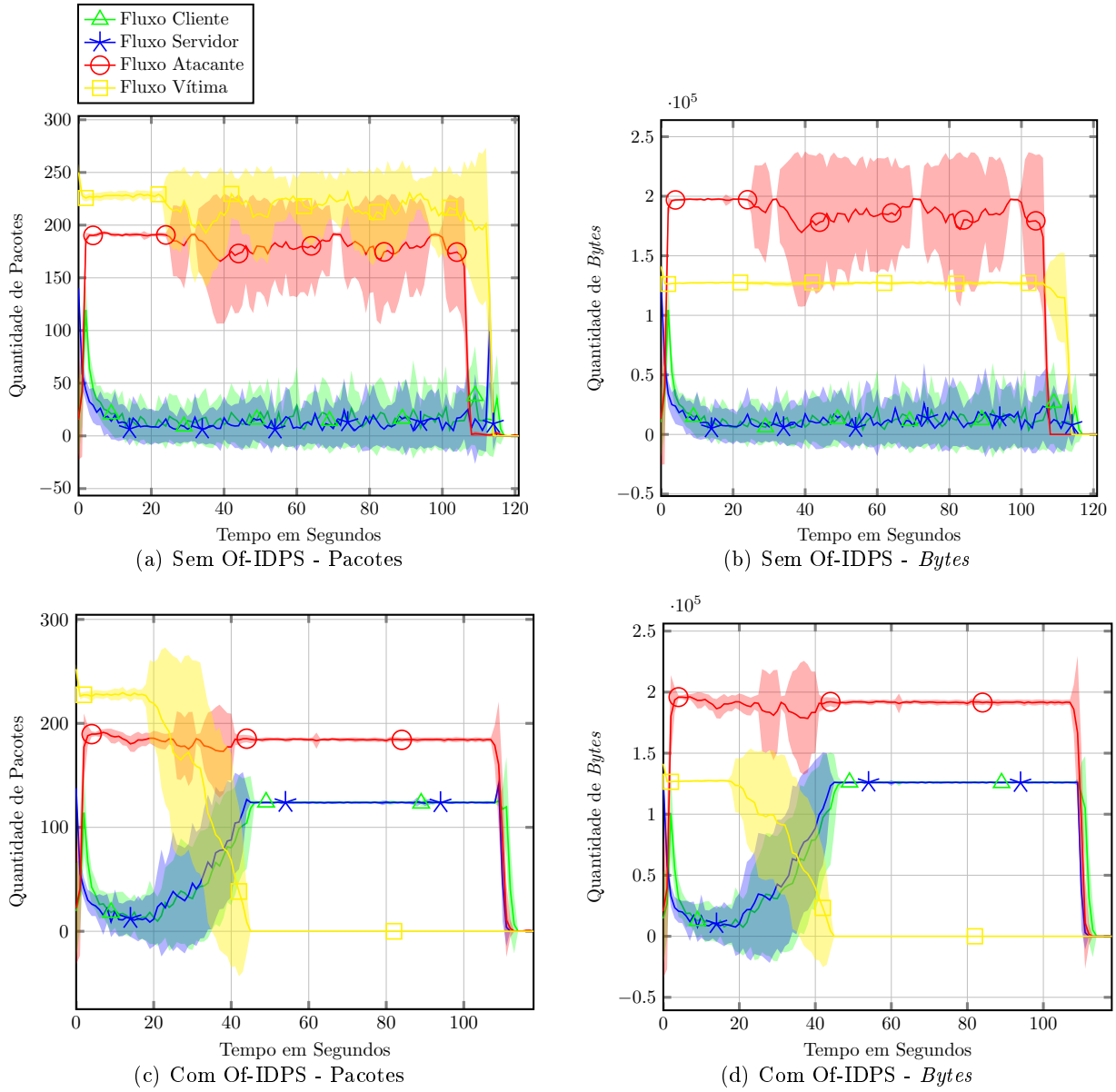
A Figura 5.13 ilustra a média e o desvio padrão de cada fluxo de rede gerado no experimento. A Tabela 5.12 apresenta os dados referentes a taxa de transferência de dados por segundo e a quantidade de *bytes* e pacotes tratados por todos os fluxos de rede presentes no referido experimento. Já a Tabela 5.13 mostra a diferença percentual entre os dados do experimento com Of-IDPS, em relação ao experimento sem Of-IDPS.

Tabela 5.12: *Experimento 2.6 - Quantidade de pacotes/bytes e média de Bps/pps.*

Of-IDPS	Fluxo	Quantidade Total		Bps	Pps
		Bytes	Pacotes		
Sem	Cliente	1.588.357	2.113	13.019	17
	Servidor	1.570.538	1.725	12.873	14
	Atacante	19.816.649	19.233	162.432	158
	Vítima	14.388.267	24.754	117.937	203
Com	Cliente	10.075.635	10.168	84.669	85
	Servidor	10.072.329	10.082	84.641	85
	Atacante	20.703.899	19.981	173.982	168
	Vítima	4.444.278	7.812	37.347	66

Examinando a Figura 5.13 nota-se que o ataque DDoS foi mitigado pelo Of-IDPS em aproximadamente 42 segundos. Ao contrário do que ocorreu no experimento anterior com DDoS (Subseção 5.1.1.2.6), percebe-se que no presente experimento, os fluxos legítimos do cliente e servidor não foram bloqueados pela reação autonômica gerada para conter o ataque. Pois, as regras autonômicas geradas pelo Of-IDPS conseguem determinar exatamente qual porta/serviço de rede está sofrendo o ataque, o que consequentemente permite criar regras para mitigar apenas os fluxos dessa porta, sem afetar os demais serviços do *host* sob ataque.

Conforme os dados apresentados na Tabela 5.12, neste experimento o acatante da rede com Of-IDPS enviou/recebeu 20 MB em 19.981 pacotes a uma taxa de 168 pps e a referida vítima tratou



(a) Sem Of-IDPS - Pacotes

(b) Sem Of-IDPS - *Bytes*

(c) Com Of-IDPS - Pacotes

(d) Com Of-IDPS - *Bytes*

Figura 5.13: *Experimento 2.6 - Média e desvio padrão dos fluxos de rede, em pacotes e bytes*

apenas 4 MB em 4.444 desses pacotes maliciosos, a uma taxa de 66 pps. Na rede sem Of-IDPS o atacante enviou 19 MB em 19.233 pacotes a 158 pps e a vítima tratou 14 MB em 24.754 pacotes a 203 pps. Então, houve respectivamente uma redução de 69% e 68% na quantidade *bytes* e pacotes enviados/recebidos, bem como 67% na taxa de pacotes por segundo, comparando a vítima da rede que utiliza Of-IDPS com a da rede sem Of-IDPS (ver Tabela 5.13). O que demostra a mitigação do ataque.

Quanto aos fluxos legítimos deste experimento, segundo os dados da Tabela 5.12, o cliente da rede com Of-IDPS enviou/recebeu 10 MB em 10.168 pacotes a uma taxa média de 85 pps e o servidor tratou 10 MB em 10.082 pacotes a 85 pps. Já no cenário da rede sem Of-IDPS o cliente enviou/recebeu em média 1.5 MB em 2.113 pacotes a 17 pps, enquanto o servidor tratou 1.5 MB em 1.725 pacotes a 14 pps. Analisando esses números nota-se que o cliente da rede com Of-IDPS teve um aumento de 534% na quantidade de *bytes* enviados/recebidos, 381% na quantidade de pacotes enviados/recebidos e 393% na taxa de pacotes por segundo, em relação a rede sem Of-IDPS. Já no caso do servidor, o aumento da rede com em relação a rede sem Of-IDPS foi de 541% na quantidade de *bytes*, 484% no total de pacotes e 499% na taxa de transferência de pacotes por segundo. Ou

Tabela 5.13: *Experimento 2.6 - Diferença em porcentagem entre experimentos sem e com Of-IDPS.*

Fluxo	Quantidade Total		Bps	Pps
	Bytes	Pacotes		
Cliente	▲ 534,34	▲ 381,10	▲ 550,33	▲ 393,23
Servidor	▲ 541,33	▲ 484,56	▲ 557,50	▲ 499,30
Atacante	▲ 4,48	▲ 3,89	▲ 7,11	▲ 6,51
Vítima	▼ 69,11	▼ 68,44	▼ 68,33	▼ 67,65

▲ Aumento percentual.
▼ Redução percentual.

seja, o Of-IDPS foi capaz de mitigar os fluxos referentes ao ataque DDoS e melhor, removendo os efeitos negativos desse ataque sobre os fluxos legítimos.

Portanto, o presente experimento demonstra que o Of-IDPS é capaz de conter ataques DDoS de inundação TCP, que estão ocorrendo em um dado serviço de rede de um *host*, sem afetar negativamente os demais serviços de rede deste mesmo *host*.

5.1.1.2.8 Experimento 2.7 - DDoS: mesmo *host* e serviço de rede, usando memória longa boa antes da curta

O objetivo deste experimento é verificar se o uso da memória longa boa do Of-IDPS permite mitigar ataques DDoS de inundação TCP ocorrendo em um dado serviço e *host*, sem influenciar negativamente os fluxos legítimos/conhecidos e que são destinados para o mesmo serviço e *host* sob ataque.

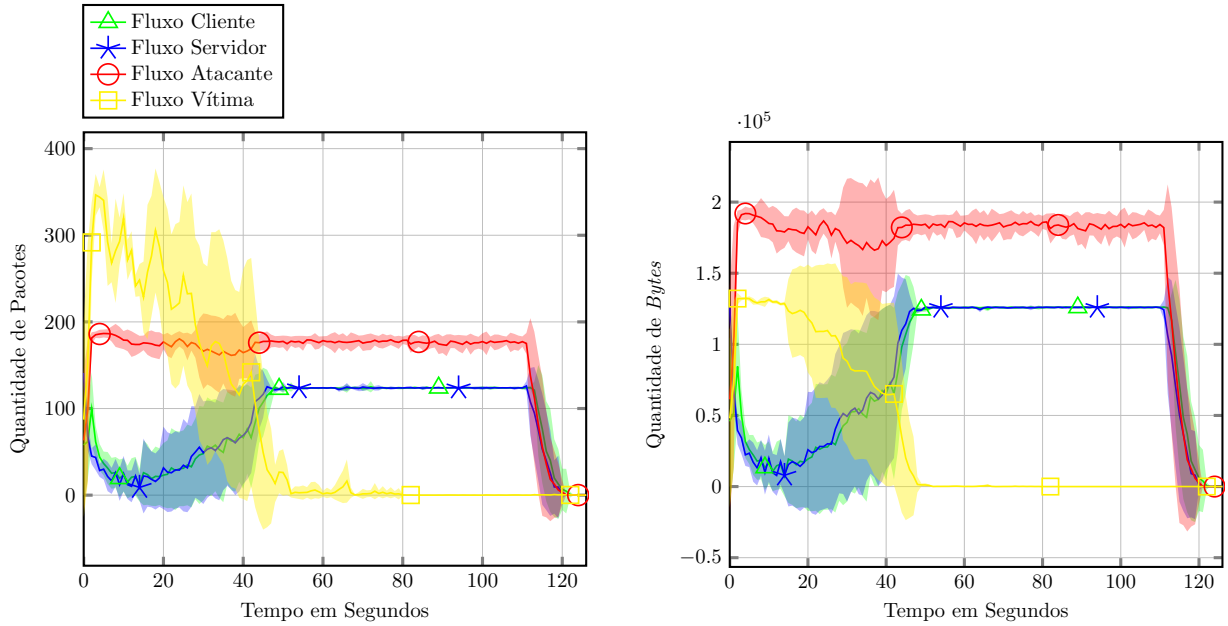
Este experimento é bem similar ao Experimento 2.5, apresentado anteriormente na Subseção 5.1.1.2.6. Consequentemente, para este experimento foi utilizado o mesmo cenário de rede do Experimento 2.5. Só que no presente experimento são empregadas em ordem as memórias sensorial, longa boa e curta. Desta forma, neste experimento é adicionada a memória longa boa com intuito de evitar que a reação autonômica, que visa mitigar o ataque, afete negativamente os fluxos de redes bem conhecidos (fluxos que normalmente são transmitidos pela rede e não estão relacionados com alertas de segurança). Neste experimento a memória longa boa analisará basicamente fluxos de redes que foram originados a partir do Wget executado no *host* 3 e destinados ao servidor HTTP Apache instalado na porta TCP/80 do *host* 1.

A Figura 5.14 ilustra a média e o desvio padrão do ataque DDoS proposto em um cenário de rede com Of-IDPS que emprega a memória longa boa. As Tabelas 5.14 e 5.15 apresentam respectivamente os dados (quantidade *bytes*/pacotes e taxa de transferência) e a diferença percentual deste experimento na rede com Of-IDPS em relação aos dados da rede sem Of-IDPS, que foram obtidos no Experimento 2.5 (Subseção 5.1.1.2.6).

Analisando a média de pacotes e *bytes* transmitidos, ilustrados na Figura 5.14, observa-se que tal como ocorreu no Experimento 2.5, os fluxos atacante/vítima são identificados e mitigados pelas regras autonômicas do Of-IDPS. Todavia, no presente experimento os fluxos referentes ao cliente/-servidor não são afetados negativamente pelas regras autonômicas do Of-IDPS, diferente do que havia ocorrido no Experimento 2.5.

Observando com mais cautela os fluxos cliente e servidor deste experimento, verifica-se que ambos são afetados inicialmente pelo ataque DDoS. Porém, depois que as memórias sensorial e curta mitigam os fluxos maliciosos, os fluxos legítimos começam a fluir normalmente (ver Figura 5.14). Ou seja, a análise feita pela memória longa boa identifica que os fluxos entre os *hosts* 3 e 4 destinados/originados da porta TCP/80 são comuns e não representam perigo a rede. Logo esses fluxos considerados legítimos pela memória longa boa não sofrem os efeitos colaterais causados pela reação autonômica gerada pela memória curta no intuito de mitigar o ataque DDoS.

Examinando os dados da Tabela 5.14, constata-se que o cliente da rede sem Of-IDPS enviou/-recebeu 1.5 MB em 2.113 pacotes a uma taxa de 13 Bps e 17 pps. Já o cliente deste experimento, utilizando a memória longa boa, enviou/recebeu 10,6 MB em 10.780 pacotes a uma taxa de 84 Bps



(a) Com Of-IDPS - Pacotes (b) Com Of-IDPS - *Bytes*

Figura 5.14: *Experimento 2.7 - Média e desvio padrão dos fluxos de rede, em pacotes e bytes*

e 85 pps. O que, em comparação com a rede sem Of-IDPS, representa respectivamente um ganho de 571% e 410% na quantidade de *bytes* e pacotes enviados/recebidos (ver Tabela 5.15). Quanto a taxa de transferência em *bytes*, essa aumentou 545% e a de pacotes aumentou 389%.

Tabela 5.14: *Experimento 2.7 - Quantidade de pacotes/bytes e média de Bps/pps.*

Of-IDPS	Fluxo	Quantidade Total		Bps	Pps
		Bytes	Pacotes		
Sem	Cliente	1.588.357	2.113	13.019	17
	Servidor	1.570.538	1.725	12.873	14
	Atacante	19.816.649	19.233	162.432	158
	Vítima	14.388.267	24.754	117.937	203
Com	Cliente	10.673.197	10.780	84.041	85
	Servidor	10.670.280	10.698	84.018	84
	Atacante	20.657.103	19.947	162.654	157
	Vítima	4.607.386	10.047	36.279	79

Se forem comparados os dados da rede com Of-IDPS do presente experimento, com os dados da rede com Of-IDPS do Experimento 2.5 (ver Tabela 5.10), o aumento foi ainda maior, pois no Experimento 2.5 o fluxo legítimo foi afetado severamente pelas contramedidas autonômicas para conter o ataque DDoS. Sendo assim, o cliente da rede com Of-IDPS, utilizando apenas as memórias sensorial e curta enviou/recebeu 0,255 MB em 377 pacotes a uma taxa de 2,1 Bps e 3 pps. Ou seja, em termos percentuais a rede utilizando a memoria longa boa aumentou, respectivamente, em 4.068% e 2.758% a quantidade de *bytes* e pacotes enviados/recebidos se comparado com o Experimento 2.5. O que demonstra, neste caso, que o uso da memória longa boa inibiu os efeitos colaterais da reação autonômica, contra o ataque DDoS, nos fluxos legítimos.

Quando se compara o fluxo do servidor da rede sem Of-IDPS com a rede com Of-IDPS deste experimento, os dados são similares ao do cliente, sendo que o servidor teve um aumento de 579% e 410% na quantidade de *bytes* e pacotes enviados/recebidos, respectivamente (ver Tabelas 5.14 e 5.15).

No que se refere aos dados dos fluxos do atacante e vítima da rede que utiliza Of-IDPS com as memórias sensorial, curta e longa boa, apresentados na Tabela 5.14. Esses são bem similares aos

Tabela 5.15: *Experimento 2.7 - Diferença em porcentagem entre experimentos sem e com Of-IDPS.*

Fluxo	Quantidade Total		Bps	Pps
	Bytes	Pacotes		
Cliente	▲ 571,96	▲ 410,05	▲ 545,51	▲ 389,97
Servidor	▲ 579,40	▲ 520,27	▲ 552,65	▲ 495,85
Atacante	▲ 4,24	▲ 3,71	▲ 0,14	▼ 0,37
Vítima	▼ 67,98	▼ 59,41	▼ 69,24	▼ 61,01

▲ Aumento percentual.
▼ Redução percentual.

dados da rede com Of-IDPS que utiliza apenas a memória sensorial e curta, do Experimento 2.5 (ver Tabela 5.10). Sendo que esses apresentam uma diferença percentual de no máximo 0,15% na quantidade de pacotes e *bytes* enviados/recebidos. Isso demonstra que o uso da memória longa boa consegue privilegiar fluxos legítimos sem influência negativamente na mitigação do ataque.

Desta forma o presente experimento demonstra que o uso da memória longa boa proposta, consegue evitar que fluxos legítimos/comuns sejam afetados pela reação autonômica criada pelo Of-IDPS para mitigar ataques agressivos, como é o caso do DDoS.

5.1.1.2.9 Experimento 2.8 - DoS: mesmo *host* e serviço de rede, usando memória longa boa antes da curta

O objetivo deste experimento é similar ao experimento anterior, com a diferença de que o ataque a ser realizado é um DoS e não um DDoS. Em outras palavras este experimento visa verificar se a memória longa boa, consegue evitar que os fluxos legítimos da rede sejam afetados negativamente pelas contramedidas geradas para conter ataques DoS de inundação de TCP.

Para este experimento é utilizado o mesmo cenário de rede do Experimento 2.4 (Subseção 5.1.1.2.5), com a diferença de que o Of-IDPS é configurado para executar em ordem as memórias sensorial, longa boa e curta, tal como no experimento anterior com DDoS.

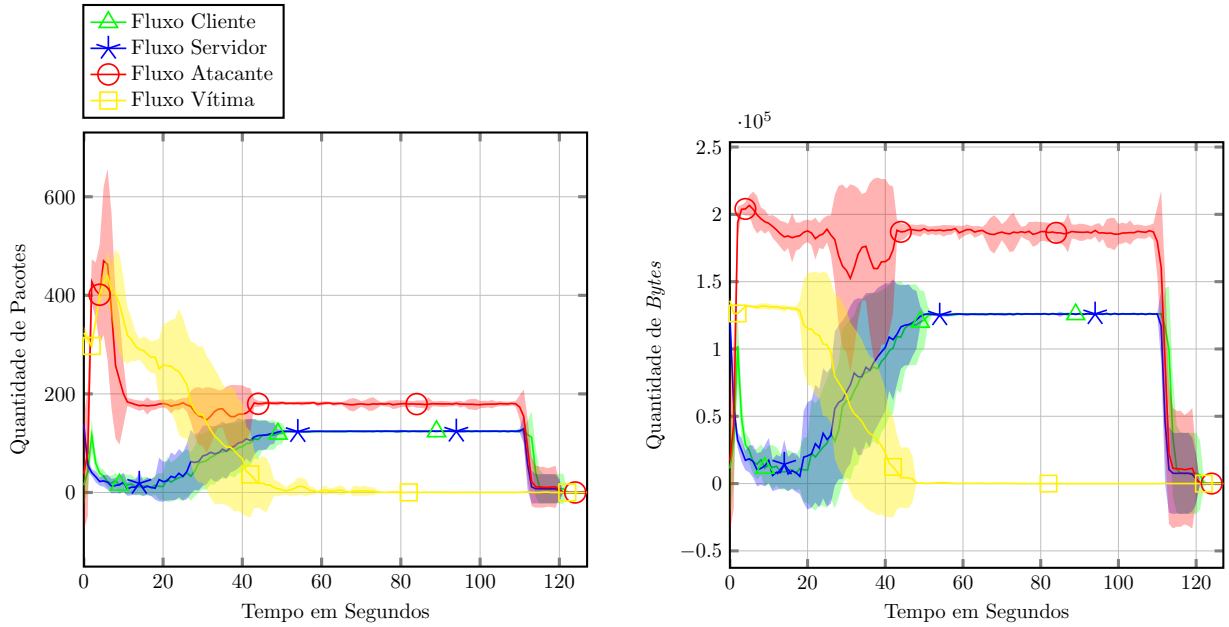
A Figura 5.15 ilustra a média e o desvio padrão dos fluxos de rede gerados por este experimento. A Tabela 5.16 apresenta os dados referentes a quantidade de *bytes* e pacotes transmitidos, bem como a taxa de transmissão de dados em Bps e pps. Já a Figura 5.17 mostra a diferença em termos percentuais dos testes em uma rede sem Of-IDPS[6] em relação a uma rede com Of-IDPS.

Portanto, a Figura 5.15 mostra que os fluxos atacante/vítima foram mitigados de forma bem similar aos experimentos com DoS apresentados anteriormente. O que demostra que o Of-IDPS é efetivo em mitigar o ataque DoS, mesmo com a adição da memória longa boa. Isso também pode ser comprovado analisando-se os dados referentes aos fluxos atacante e vítima das Tabelas 5.16 e 5.17.

Tabela 5.16: *Experimento 2.8 - Quantidade de pacotes/bytes e média de Bps/pps.*

Of-IDPS	Fluxo	Quantidade Total		Bps	Pps
		Bytes	Pacotes		
Sem	Cliente	1.552.369	2.153	12.519	18
	Servidor	1.531.660	1.663	12.352	15
	Atacante	20.316.751	19.784	163.845	165
	Vítima	14.563.366	30.332	117.447	245
Com	Cliente	10.694.216	10.740	83.549	84
	Servidor	10.693.222	10.720	83.541	84
	Atacante	20.539.005	21.324	160.461	167
	Vítima	4.103.932	10.371	32.062	81

[6] São os mesmo dados da rede sem Of-IDPS do Experimento 2.4 na Tabela 5.8



(a) Com Of-IDPS - Pacotes (b) Com Of-IDPS - *Bytes*

Figura 5.15: *Experimento 2.8 - Média e desvio padrão dos fluxos de rede, em pacotes e bytes*

Tabela 5.17: *Experimento 2.8 - Diferença em porcentagem entre experimentos sem e com Of-IDPS.*

Fluxo	Quantidade Total		Bps	Pps
	Bytes	Pacotes		
Cliente	▲ 588,90	▲ 398,92	▲ 567,37	▲ 356,04
Servidor	▲ 598,15	▲ 544,53	▲ 576,33	▲ 474,03
Atacante	▲ 1,09	▲ 7,79	▼ 2,07	▲ 1,05
Vítima	▼ 71,82	▼ 65,81	▼ 72,70	▼ 66,88

▲ Aumento percentual.
▼ Redução percentual.

Quanto aos fluxos considerados legítimos (cliente/servidor) a Figura 5.15 mostra que esses fluxos não apresentam mais perturbações, tão logo os fluxos maliciosos sejam mitigados pelas regras autonômicas das memórias sensorial e curta.

Por exemplo, comparando os dados do Experimento 2.4 que usa o Of-IDPS, mas sem a memória longa boa (Tabela 5.8), com os dados do presente experimento, que usa a memória longa boa (Tabela 5.16). Contata-se que no Experimento 2.4 o cliente enviou/recebeu 3,1 MB em 3.388 pacotes a uma taxa de 26 Bps e 28 pps. Já no presente experimento, o cliente tratou 10,6 MB em 10.740 pacotes com uma taxa de 83 Bps e 84 pps. Desta forma, o cliente do experimento com a memória longa boa obteve um aumento de 239% na quantidade de *bytes* enviados/recebidos, 217% na quantidade total de pacotes, 218% na taxa de Bps e 197% na taxa de pps, em relação ao experimento que não utiliza a memória longa boa (o servidor apresenta dados semelhantes a esses do cliente). Portanto, essa comparação entre os experimentos, ajudam a demonstrar que a memória longa boa consegue evitar que fluxos legítimos sejam afetados pela reação autonômica que visa mitigar ataques DoS.

Agora, comparando a rede sem Of-IDPS com a rede que utiliza Of-IDPS com memória longa boa (Tabela 5.16). Observa-se que na rede sem Of-IDPS o cliente sofreu com os efeitos do ataque DoS, pois enviou/recebeu apenas 1.5 MB em 2.153 pacotes a uma taxa de 12 Bps e 18 pps. Assim, comparando ambos cenários de rede, constata-se que houve um aumento de 588% na quantidade de *bytes* tratados, 398% na quantidade total de pacotes, 567% na taxa de Bps e 356% pps, da rede com Of-IDPS em relação a rede sem Of-IDPS. O percentual de aumento do servidor é similar ao do cliente e pode ser visto na Tabela 5.17. Esses resultados demonstram que a memória longa boa somada a reação autonômica das outras memórias consegue mitigar problemas de segurança que

estejam ocorrendo na rede e privilegiar os fluxos identificados como legítimos.

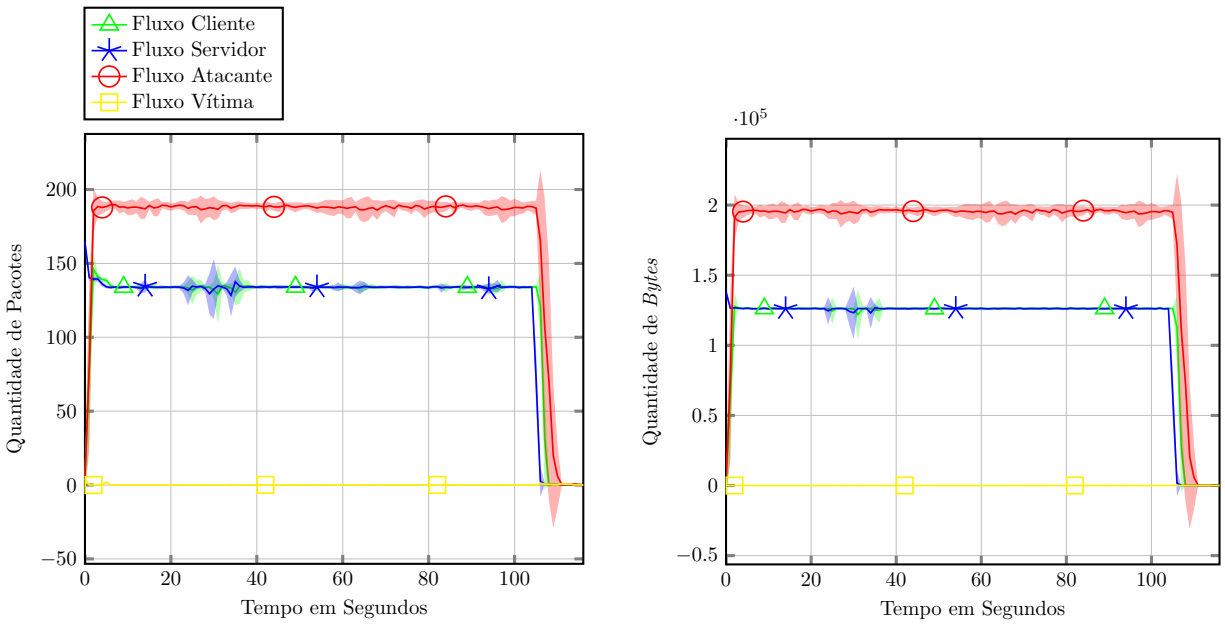
Portanto, o experimento demonstra que o Of-IDPS utilizando a memória longa boa é capaz de mitigar um ataque DoS de inundação TCP destinado a um dado *host*/serviço, sem interferir em fluxos legítimos que estejam sendo destinados ao mesmo *host*/serviço que está sob ataque.

5.1.1.2.10 Experimento 2.9 - DoS: mesmo *host* e serviço de rede, usando memória longa boa e ruim

O objetivo deste experimento é avaliar se a memória longa ruim, somada a todas as outras memórias do Of-IDPS conseguem mitigar ataques DoS de inundação de TCP, sem afetar fluxos legítimos.

Este experimento utiliza o mesmo cenário de rede e configuração do Experimento 2.8 (Subseção 5.1.1.2.9), com a diferença de empregar em ordem as memórias sensorial, longa boa, curta e longa ruim. Assim, as memórias sensorial, curta e longa boa são configuradas tal como no Experimento 2.8. Já a memória longa ruim é adicionada ao Of-IDPS e irá analisar os alertas de segurança gerados de ataques originados do *host* 4 e destinados a porta TCP/80 do *host* 1. Desta forma, espera-se que o Of-IDPS, utilizando principalmente as memórias longas, seja capaz de avaliar e aprender com o histórico de fluxos legítimos e maliciosos da rede, para então mitigar ataques ocorrendo do *host* 4 para o *host* 1 na porta TCP/80, sem interferir na transferência de dados idônea do *host* 3 para a porta TCP/80 do mesmo *host* 1.

A Figura 5.16 ilustra a média e o desvio padrão dos fluxos de rede gerados neste experimento. As Tabelas 5.18 e 5.19 apresentam, respectivamente a quantidade de *bytes*/pacotes enviados recebidos e a diferença em termos percentuais de uma rede utilizando Of-IDPS, com todas as memórias do experimento, em relação a uma rede sem Of-IDPS.



(a) Com Of-IDPS - Pacotes (b) Com Of-IDPS - *Bytes*

Figura 5.16: *Experimento 2.9 - Média e desvio padrão dos fluxos de rede, em pacotes e bytes*

Analisando os fluxos apresentados na Figura 5.16, observa-se que a memória longa ruim, foi capaz de mitigar completamente os efeitos do ataque sobre a vítima. Uma vez que, o fluxo da vítima permanece em zero (0) durante praticamente todo o ataque. Já os pacotes do atacante são enviados para a rede, todavia esses não ultrapassam a porta do *switch* OpenFlow, ou seja, são barrados pelas regras autonômicas geradas pelas memórias sensorial, curta e longa ruim do Of-IDPS. Conforme pode ser visto na Tabela 5.18 a vítima da rede com Of-IDPS recebeu apenas 655

Tabela 5.18: *Experimento 2.9 - Quantidade de pacotes/bytes e média de Bps/pps.*

Of-IDPS	Fluxo	Quantidade Total		Bps	Pps
		Bytes	Pacotes		
Sem	Cliente	1.552.369	2.153	12.519	18
	Servidor	1.531.660	1.663	12.352	15
	Atacante	20.316.751	19.784	163.845	165
	Vítima	14.563.366	30.332	117.447	245
Com	Cliente	13.321.967	14.187	113.863	121
	Servidor	13.321.487	14.178	113.859	121
	Atacante	20.801.132	20.024	177.787	171
	Vítima	655	15	6	0

Tabela 5.19: *Experimento 2.9 - Diferença em porcentagem entre experimentos sem e com Of-IDPS.*

Fluxo	Quantidade Total		Bps	Pps
	Bytes	Pacotes		
Cliente	▲ 758,17	▲ 559,01	▲ 809,51	▲ 559,01
Servidor	▲ 769,74	▲ 752,42	▲ 821,78	▲ 730,56
Atacante	▲ 2,38	▲ 1,21	▲ 8,51	▲ 3,81
Vítima	▼ 99,996	▼ 99,95	▼ 99,995	▼ 99,95

▲ Aumento percentual.
▼ Redução percentual.

bytes em 15 pacotes[7]. Já na rede sem Of-IDPS, a vítima recebeu 14 MB provindos de 30.332 pacotes maliciosos. Então, a vítima da rede com Of-IDPS obteve uma redução de 99,996% na quantidade de *bytes* enviados/recebidos e 99,95% na quantidade de pacotes maliciosos enviados/recebidos em comparação a vítima da rede sem Of-IDPS (ver Tabela 5.19).

Ainda analisando a Figura 5.16, nota-se que os fluxos legítimos (cliente/servidor) não são afetados pelo ataque DoS e nem pela reação autonômica, que impede o ataque. Isso pode ser melhor constatado analisando os dados da Tabela 5.18, a qual por exemplo, mostra que o cliente da rede sem Of-IDPS enviou/recebeu 1.5 MB em 2.153 pacotes de rede, ou seja, sofreu os efeitos do ataque DoS. Já o cliente protegido pela rede com Of-IDPS e todas as suas memórias, enviou/recebeu 13 MB em 14.187 pacotes de rede. Desta maneira, o cliente da rede com Of-IDPS obteve respectivamente um aumento de 758% e 559% na quantidade de *bytes* e pacotes enviados recebidos, em relação ao cliente da rede sem proteção. No que se refere ao servidor, esse teve respectivamente um aumento de 769% e 752% na quantidade de *bytes* e pacotes enviados/recebidos na rede com Of-IDPS em relação a rede sem proteção (ver Tabela 5.19).

Portando este experimento demonstra que o uso do Of-IDPS combinando todas as suas memórias (sensorial, curta, longa boa e longa ruim) pode ajudar a mitigar ataques DoS de forma efetiva, bem como evitar que fluxos legítimos e bem conhecidos sejam afetados pelo ataque ou pela reação autonômica criada pela própria arquitetura proposta para conter o ataque.

5.1.1.3 Experimento 3 - Reação contra diversos tipos de ataques

O objetivo deste experimento é analisar a reação autonômica do Of-IDPS quando esse é submetido a diversos tipos de pacotes maliciosos que representam diferentes ataques, que são ainda destinados a várias portas e serviços de rede.

O cenário de rede do experimento é representado pela Figura 5.1, sendo que o atacante é o *host* 4 e a vítima é o *host* 1. Durante o experimento o atacante envia diversos pacotes de rede suspeitos usando a ferramenta IDSWakeup (IDSwakeup, 2016).

[7] Os *bytes* e pacotes do fluxo da vítima são referentes a pacotes ICMP gerados pelo comando ping e que são utilizados para marcar o inicio e o final desses testes. Além de pacotes ARP que são comumente enviados pela rede.

O IDSWakeup é normalmente utilizado, por administradores de rede, para verificar se NIDS estão gerando alertas quando expostos a ataques. Tais alertas são gerados, pois o IDSWakeup envia vários pacotes de rede[8] que combinam com as assinaturas de ataques normalmente conhecidos por NIDS, tal como:

- Teardrop: Ataque DDoS envolvendo o envio de pacotes fragmentados para a vítima;
- LAND (*Local Area Network Denial*): Ataque DoS que consiste em enviar pacotes com o mesmo endereço de origem e destino da vítima;
- Get phf: Explora vulnerabilidades de programas CGI (*Common Gateway Interface*) e pode permitir a execução de comandos remotamente;
- Ping of Death: Ataque que envia pacotes ICMP mal formados através do comando *ping*, visando comprometer os recursos da vítima;
- NewTear: Ataque no estilo Teardrop;
- SMBnegprots: Ataque DoS que utiliza mensagens SMB (*Server Message Block*);
- SMTP expn root: Tentativa de descobrir os e-mails associados com o usuário *root* no Sendmail;
- FTP cwd root: Tentativa de escalar um usuário para *root* em servidores FTP;
- Msadcs: Ataque que permite executar comandos remotamente no MDAC (*Microsoft Data Access Components*);
- DoS Chargen: Ataque contra algumas versões do sistema operacional Windows, no qual são enviados pacotes *broadcast* para a porta UDP/19, conhecida como *chargen port*;
- DoS Snork: Ataque que envia pacotes com dados randômicos e com endereços de origem falsos para a porta UDP/135;
- Outros: também são enviados pacotes de rede HTTP, FTP, *telnet*, *rlogin*, ICMP e TCP, que representam ações maliciosas[9] executadas por *hackers* (IDSwakeup, 2016).

Durante a execução do IDSWakeup o Snort emite 199 alertas que representam 33 tipos de incidentes contra a segurança da rede. A Tabela 5.20 apresenta os 33 tipos de alertas gerados pelo Snort durante a execução do experimento com o IDSWakeup. Nessa tabela, a primeira coluna apresenta os códigos de identificação do Snort para cada tipo de alerta gerado. A segunda coluna faz uma descrição geral do incidente de segurança para grupos de alertas similares. Por exemplo, os tipos de alertas com identificação 223, 244, 245, 246 e 249 estão relacionados com ataques DoS e DDoS. Já os alertas 1139, 1142, 1156 fazem referência a incidentes contra a segurança de servidores ou aplicações Web. Uma descrição mais detalhada, de cada alerta gerado pelo Snort, pode ser obtida submetendo-se o código de identificação do alerta (primeira coluna da Tabela 5.20) a um serviço de busca disponível em https://www.snort.org/rule_docs/.

Sendo assim, neste experimento o atacante, utilizando o IDSWakeup, irá enviar vários pacotes maliciosos para a vítima. Por sua vez o Of-IDPS identifica tais pacotes, utilizando o sensor Snort. Na sequencia, o Of-IDPS empregando as memórias sensorial, curta, longa boa e longa ruim, analisa os alertas gerados e cria regras de segurança para conter os referidos pacotes maliciosos. O experimento foi repetido 30 vezes e o IDSWakeup é executado duas vezes consecutivas em cada experimento.

A Figura 5.17 apresenta os fluxos da vítima (linha azul marcada com um círculo) e atacante (linha vermelha, pontilhada e marcada com uma estrela), bem como o momento em que termina

[8] Os pacotes são referentes a diferentes tipos de mensagens ICMP e no caso dos protocolos TCP e UDP são destinados a várias portas de rede ou serviços.

[9] Como por exemplo, tentativa de identificação de recursos da rede, tentativa de escalar para o usuário *root*, execução indevida de comandos remotos, etc.

Tabela 5.20: *Alertas gerados pelo Snort durante a execução do IDSWakeup.*

Snort ID*	Breve Descrição dos Alertas
223	Alertas relacionados com ataques DoS e DDoS.
244	
245	
246	
249	
384	Alertas relacionados com ataques que utilizam vários tipos de datagramas ICMP.
386	
388	
396	
397	
401	
402	
403	
406	
436	
451	
472	
525	Alerta gerado por ataques que exploram vulnerabilidades do protocolo ou implementações do UDP.
624	Alerta gerado por ataques que exploram vulnerabilidades do protocolo ou implementações do TCP.
659	Alertas gerados por ataques que exploram vulnerabilidades do SNMP ou de servidores de e-mail.
663	
714	Alerta gerado por ataques que podem ser identificados no uso do Telnet.
1113	Alerta gerado durante ataques Directory Transversal.
1139	Alertas gerados durante ataques a servidores ou aplicações Web.
1142	
1156	
1411	Alertas gerados durante ataques que exploram o protocolo SNMP.
1419	
1672	Alertas gerados durante ataques que exploram vulnerabilidades de implementações de servidores FTP.
1971	
3441	
336	
361	

* - Uma descrição mais detalhada de cada alerta pode ser obtida em https://www.snort.org/rule_docs/.

a primeira execução do IDSWakeup e inicia a segunda execução (barra/linha laranja marcada com um triângulo). A Figura 5.18 mostra separadamente o desvio padrão de vítima e atacante dos experimentos sem e com Of-IDPS. A Tabela 5.21 apresenta a quantidade de pacotes enviados e recebidos pelos fluxos gerados por atacante e vítima, bem como a taxa de pacotes por segundo transmitida em experimentos com e sem Of-IDPS. A Tabela 5.22 mostra a diferença em porcentagem dos dados do experimento que utiliza Of-IDPS em relação ao experimento sem Of-IDPS.

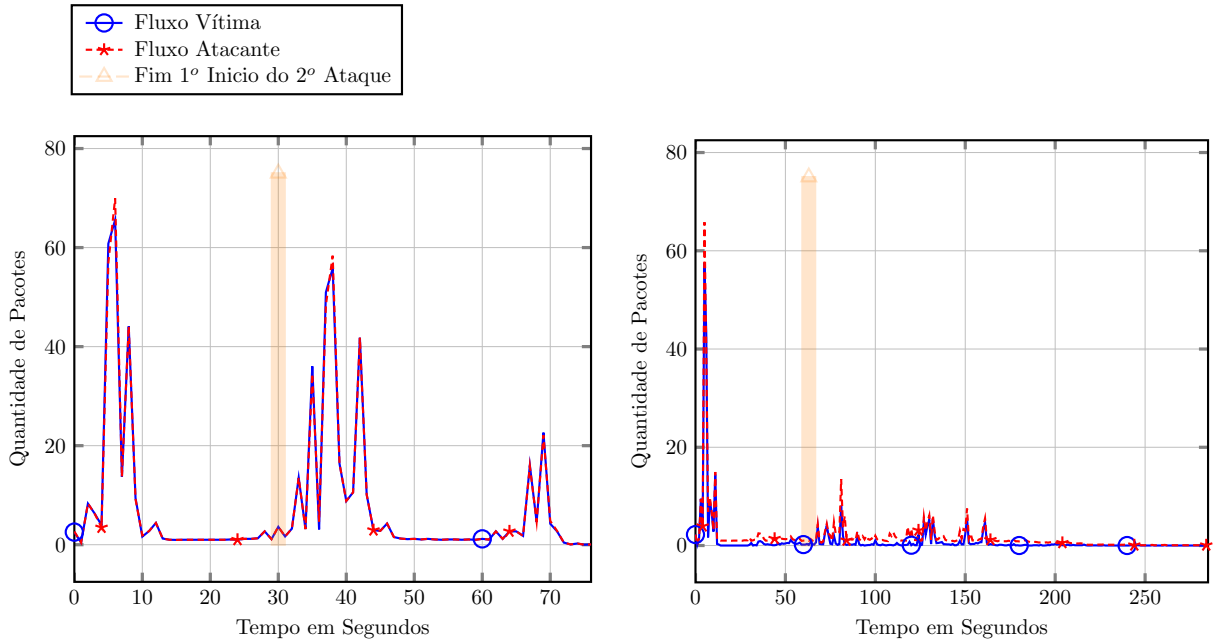


Figura 5.17: *Fluxo de pacotes durante testes com IDSwakeup.*

Tabela 5.21: *Experimento 3 - Diferença em porcentagem entre experimentos sem e com Of-IDPS.*

Of-IDPS	Fluxo	Quantidade Total Pacotes	Pps
Sem	Atacante	591	8
	Vítima	591	8
Com	Atacante	484	2
	Vítima	261	1

Tabela 5.22: *Experimento 3 - Diferença em porcentagem entre experimentos sem e com Of-IDPS.*

Fluxo	Quantidade Total Pacotes	pps
Atacante	▼ 18,08	▼ 77,94
Vítima	▼ 55,93	▼ 88,13

▲ Aumento percentual.
▼ Redução percentual.

No experimento sem o uso do Of-IDPS (Figura 5.17-a), nota-se dois grandes picos de pacotes no fluxo da vítima, sendo um pico de 65 pacotes e outro de 55, que representam respectivamente a execução do primeiro e segundo ataque causados pelas execuções do IDSWakeup.

Agora, analisando os fluxos do cliente da Figura 5.17-b, que representa o experimento com Of-IDPS, observa-se que houve um primeiro grande pico de 57 pacotes, causado pelo primeiro ataque. Depois no segundo ataque, o pico não passou de 8 pacotes. Isso ocorreu porque durante a primeira execução do IDSWakeup o Of-IDPS ainda está percebendo o ataque para então reagir. Já durante a segunda execução do IDSWakeup o sistema já implantou autonomicamente regras de segurança que bloqueiam ou suavizam o ataque, o que também pode ser visto na Figura 5.18.

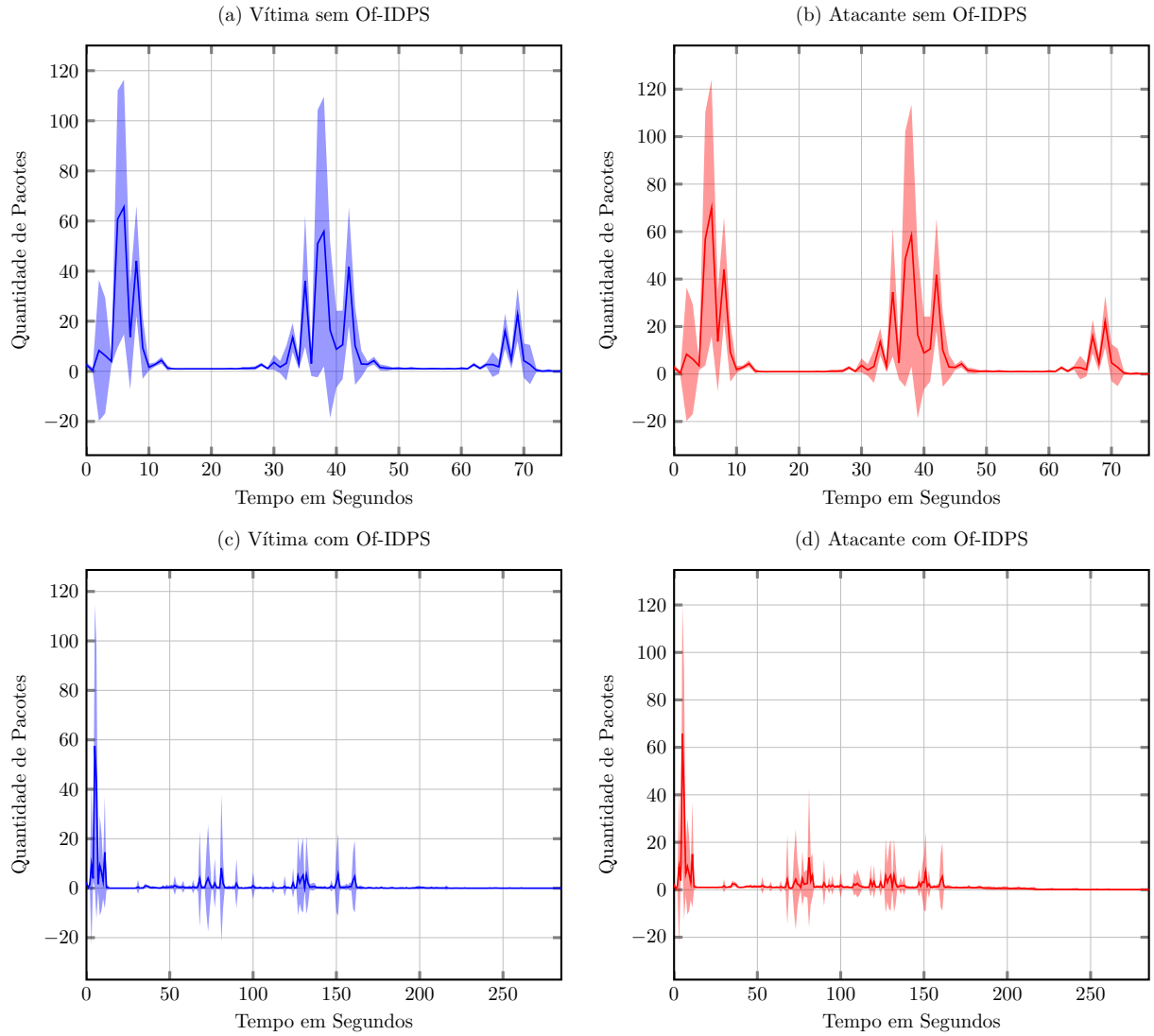


Figura 5.18: *Desvio padrão dos fluxos de pacotes durante testes com IDSwakeup.*

Comparando os dados dos experimentos com e sem Of-IDPS apresentados nas Tabelas 5.21 e 5.22, observa-se que o Of-IDPS consegue mitigar os pacotes gerados pelo IDSWakeup. Pois, na rede sem Of-IDPS tanto atacante quanto a vítima enviam/recebem 591 pacotes a uma taxa de 8 pps. Na rede com Of-IDPS o atacante envia/recebe 484 pacotes a uma taxa de 2 pps, já o cliente dessa rede trata 261 pacotes a uma taxa e 1 pps. Ou seja, comparando em termos percentuais os dados das redes com e sem Of-IDPS (Tabela 5.22), nota-se que houve uma redução de 18% e 77%, respectivamente na quantidade de pacotes enviados/recebidos e na taxa de pps no fluxo do atacante. Já no fluxo da vítima a redução foi de 55% na quantidade de pacotes enviados/recebidos e 88% na taxa de pps.

Também nos experimentos sem o Of-IDPS, os ataques duravam no melhor caso 68 segundos e no pior caso 76 segundos, já nos experimentos com o Of-IDPS o tempo de ataque no melhor caso foi de 129 segundos e no pior caso 285 segundos. Ou seja, o Of-IDPS influenciou o tempo do ataque reduzindo a taxa de transferência de dados o que atrasou o ataque em 89% no melhor caso e 276% no pior caso.

Portanto, o experimento com o IDSWakeup mostra que a integração NIDS e OpenFlow por meio do Of-IDPS viabiliza um mecanismo para monitorar a rede, identificar problemas e executar ações que eliminem ou amenizem problemas de segurança em redes locais.

5.1.1.4 Experimento 4 - Uso do Of-IDPS em uma rede real em produção

O objetivo deste experimento é verificar a reação autonômica do Of-IDPS frente às adversidades impostas por um ambiente de rede real em produção. Mais especificamente os objetivos são: (1) verificar a reação do Of-IDPS frente a outros problemas de segurança, além dos já avaliados nos experimentos anteriores; (2) analisar se as respostas autonômicas contra problemas de segurança não afetam negativamente *hosts*/serviços de redes legítimos; (3) avaliar a eficácia das memórias longa boa e ruim em utilizar o histórico da rede para identificar os principais *hosts*/serviços e ataques da LAN.

Desta forma, o Of-IDPS foi instalado na LAN do GT-EWS (Grupo de Trabalho *Early Warning System*) da RNP (Rede Nacional de Ensino e Pesquisa). Tal LAN fornece infraestrutura para o funcionamento de um sistema de alertas antecipados fundamentado principalmente em mensagens de segurança postadas em redes sociais (Santos *et al.*, 2013). O sistema de alerta antecipado do GT-EWS pode ser acessado via Internet em http://gtews.cm.utfpr.edu.br/ews/ e será melhor detalhado na Seção 5.2.1.1.

A estrutura de rede do GT-EWS é apresentada na Figura 5.19, sendo essa composta basicamente por:

- Um Controlador OpenFlow, no qual está instalado o Of-IDPS;
- Um *host* coletor, responsável por obter postagens no Twitter e Facebook. Esse coleta mensagens escritas em inglês e relacionadas com cibersegurança;
- Um servidor Xen[10] que fornece um ambiente para se manter e gerenciar máquinas virtuais.

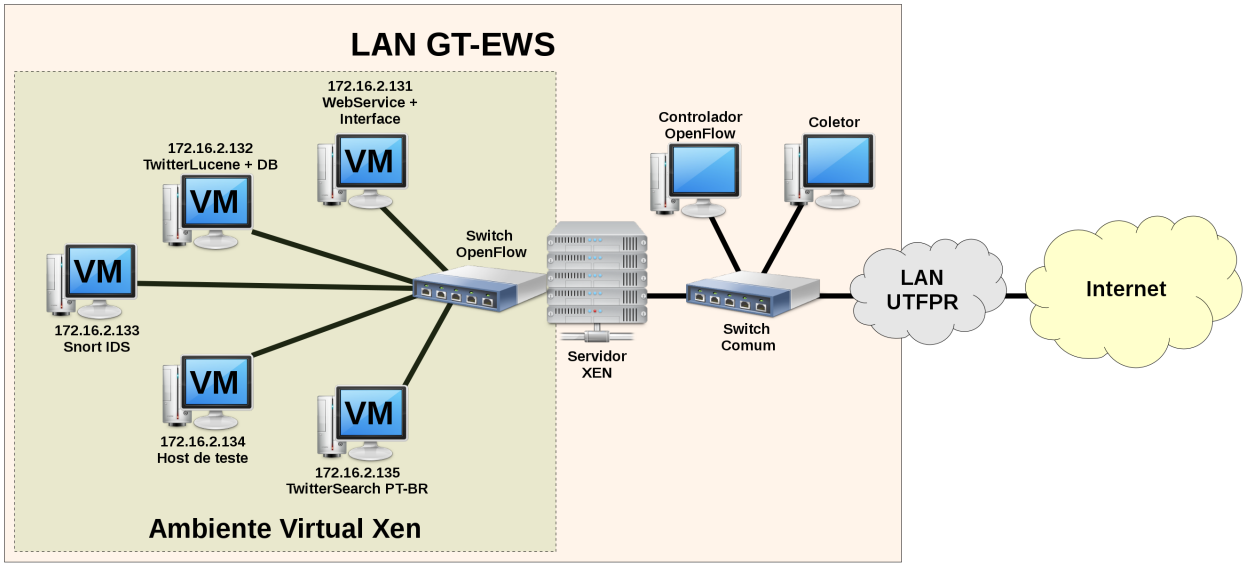


Figura 5.19: *Estrutura de rede do GT-EWS.*

Assim, o servidor Xen mantém máquinas virtuais (do inglês VM (*Virtual Machine*)), também ilustradas na Figura 5.19 e que dão suporte aos serviços do GT-EWS e do próprio Of-IDPS. Sendo tais máquinas virtuais:

- TwitterSearch PT-BR: Responsável por obter *tweets* postados em português (idioma) e que sejam relacionados com cibersegurança. Essa VM possui o IP 172.16.2.135;
- Host de teste: *Host* utilizado para alguns experimentos, tal como testes com novos tipos de coletores/sensores para o GT-EWS. Essa VM possui o IP 172.16.2.134;

[10] http://xenserver.org/ - acessado em 03 de dezembro de 2015.

- Snort IDS: Monitora a rede do GT-EWS em busca de problemas de segurança. Serve de sensor para o Of-IDPS. Essa VM possui o IP 172.16.2.133;
- TwitterLucene+DB: É o núcleo de processamento de alertas antecipados e também mantém um banco de dados que armazena informações de sensores, alertas de segurança gerados, dentre outras informações do GT-EWS. Essa VM possui o IP 172.16.2.132;
- WebService+Interface: Mantém um *Web service* que é responsável por interconectar as diferentes partes que compõem o sistema de alertas antecipados do GT-EWS (sensores, núcleo de processamento, etc). Também fornece uma interface Web que permite a visualização dos alertas de segurança gerados pelo sistema.

A LAN do GT-EWS ainda está conectada na LAN da UTFPR, que por sua vez está ligada à Internet. Então, é função do Of-IDPS proteger a LAN contra ciberameaças provindas da própria LAN do GT-EWS, da LAN da UTFPR e da Internet (ver Figura 5.19).

Para o presente experimento o Of-IDPS ficou em execução na LAN do GT-EWS durante 6 meses. Neste período o módulo Monitor (Figura 4.6) identificou 44 tipos de ciberameaças, que deram origem a 15.098 alertas de segurança. Os 44 tipos de ciberameaças obtidos podem ser vistos na Tabela 5.23. Sendo que a primeira coluna da tabela (Snort ID) apresenta um número que identifica o alerta gerado pelo Snort e a segunda coluna apresenta uma breve descrição do alerta de segurança.

Todo esse histórico de alertas de segurança é processado pela metodologia da memória longa ruim, o que resulta nas duas regras autonômicas que são apresentadas na tabela da Figura 5.20. Para uma melhor visualização do comportamento do Of-IDPS na rede, a equipe do GT-EWS desenvolveu uma interface Web que permite ver em tempo de execução as regras de segurança presentes nas memórias do Of-IDPS. Um exemplo dessa interface é apresentado na Figura 5.20. Basicamente a interface Web possui um menu lateral esquerdo que dá acesso a cada uma das memórias do Of-IDPS e uma tabela para visualização das regras de segurança geradas pela referente memória. Então, cada linha da tabela representa uma regra de segurança gerada pelo Of-IDPS e essas linhas são formadas pelos campos:

- *Priority*: Prioridade de segurança;
- *Descrip.*: Identificação do problema que originou a regra;
- *Source*: IP de origem;
- *Dest.*: IP de destino;
- *Proto.*: Protocolo;
- *T.Source*: Porta de origem;
- *T.Dest*: Porta de destino;
- *Support*: Suporte;
- *Life*: Contador de vida da regra (só é utilizado pela memória curta).

Desta forma, a primeira regra da memória longa ruim (primeira linha da tabela na Figura 5.20) bloqueia pacotes originados de 172.16.255.201, destinados a 172.16.2.131, usando o protocolo TCP, originados da porta cliente 54585 e destinados a porta 8080. Essa regra foi formada por alertas que representam mais de um tipo de problema de segurança, por isso a generalização (*none*) do campo identificação (*Descrip.*). No Of-IDPS pacotes com alta (*high*) prioridade de segurança são bloqueados (Santos *et al.*, 2014). A segunda regra faz basicamente o mesmo que a primeira, mas generalizando a porta de origem (*) e especificando o tipo de ataque (*Snort Alert [1:100000122:1]*). Ambas regras condizem com o cenário de rede analisado, pois devido à estrutura da LAN da

Tabela 5.23: *Alertas de segurança obtidos na LAN do GT-EWS durante Experimento 4.*

Snort ID*	Breve Descrição dos Alertas
254	Alertas relacionados com ataques que exploram vulnerabilidades de DNS.
1616	
366	Alertas relacionados com pacotes ICMP suspeitos.
368	
384	
399	
401	
402	
408	
449	
472	
485	
486	
503	Alertas gerados por ataques que exploram vulnerabilidades de pacotes de rede ou dispositivos.
504	
527	
528	
2189	
1112	Alertas gerados por ataques Directory Transversal.
1113	
1418	Alertas gerados durante ataques que exploram o protocolo SNMP.
1420	
1421	
1892	
1893	
1062	Alertas gerados durante ataques que visam identificar recursos e vulnerabilidades da rede.
498	
1333	
1497	
1882	
2381	
882	Alertas gerados durante ataques a servidores ou aplicações Web.
886	
1010	
1054	
1564	
1653	
1852	
1861	
1142	
1156	
1201	
100000118	
100000122	

* - Uma descrição mais detalhada de cada alerta pode ser obtida em https://www.snort.org/rule_docs/.

UTFPR, todo acesso provindo da Internet chega na LAN do GT-EWS por intermédio do *host* 172.16.255.201. Também, a maior parte dos acessos da Internet são destinados ao servidor HTTP (*host* 172.16.2.131). Então, na LAN do GT-EWS, a memória longa ruim aprendeu que muitos ataques ocorrem partindo do *host* 172.16.255.201, para 172.16.2.131 e essas ciberameaças atentam principalmente contra vulnerabilidades do servidor HTTP Apache (*Snort Alert [1:100000122:1]*), na porta TCP/8080 e por isso devem ser bloqueadas.

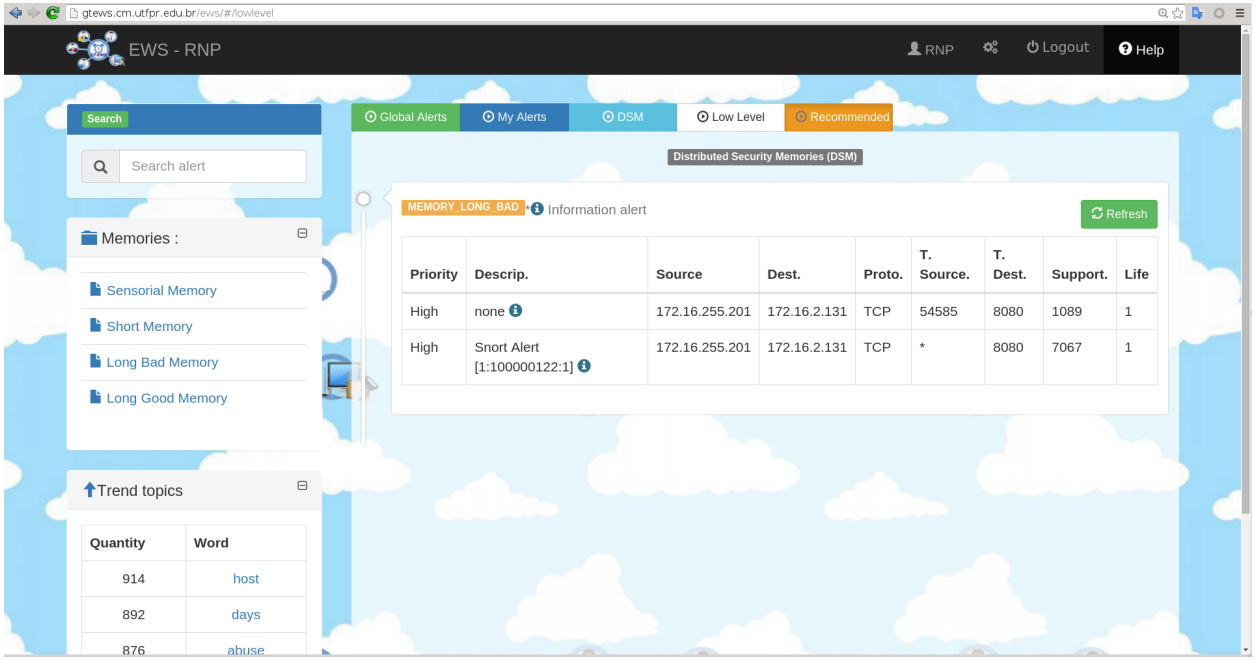


Figura 5.20: *Interface Web do GT-EWS apresentando o processamento da memória longa ruim durante Experimento 4.*

Neste mesmo período de 6 meses, foram também instalados nos *switches* OpenFlow, 10.205.790 fluxos de rede considerados idôneos (base Fluxos Normais da Figura 4.6). A análise desses fluxos pela memória longa boa pode ser vista na Figura 5.21. A primeira e a penúltima regra indicam que é comum a criação de fluxos de rede entre os *hosts* 172.16.2.131 e 172.16.255.201, usando a porta TCP/8080. Na interface Web, o campo *Priority* na memória boa é marcado como *None*, simbolizando que não há prioridade/risco de segurança para os pacotes que combinam com essas regras. Já o campo identificação (*Descrip.*) é preenchido com *good*, representando que a regra foi gerada pela memória longa boa. As demais regras demonstram que é ordinário o acesso dos *hosts* 172.16.2.131 (Web *Service* e servidor HTTP) e 192.168.2.110 (Controlador OpenFlow) ao *host* 172.16.2.132 na porta TCP/5432 (banco de dados). As regras criadas pela memória longa boa, também são condizentes com o uso da LAN do GT-EWS, pois nessa rede é muito comum que o *host* 172.16.255.201 acesse o servidor HTTP, que por sua vez, acessa o *host* do banco de dados. Já o controlador OpenFlow, que representa o núcleo do Of-IDPS, armazena as regras de segurança geradas por esse, no *host* do banco de dados, para que sejam visualizadas posteriormente no servidor HTTP.

Para possibilitar uma investigação mais minuciosa a respeito da segurança da rede do GT-EWS durante os testes com o Of-IDPS, todos os *hosts* foram monitorados adicionalmente pelas ferramentas OSSEC-HIDS[11] e CACTI[12]. Todos os dados obtidos a respeito da segurança da LAN passaram ainda pela investigação constante dos autores desse trabalho e por mais dois especialistas em cibersegurança. Todavia, em nenhum momento foi constatado o comprometimento da segurança da LAN do GT-EWS, destacando que a abordagem proposta consegue efetivamente mitigar ciberameaças sem afetar negativamente os serviços legítimos da LAN.

[11] http://ossec.github.io/ - acessado em 03 de dezembro de 2015.
[12] http://www.cacti.net/ - acessado em 03 de dezembro de 2015.

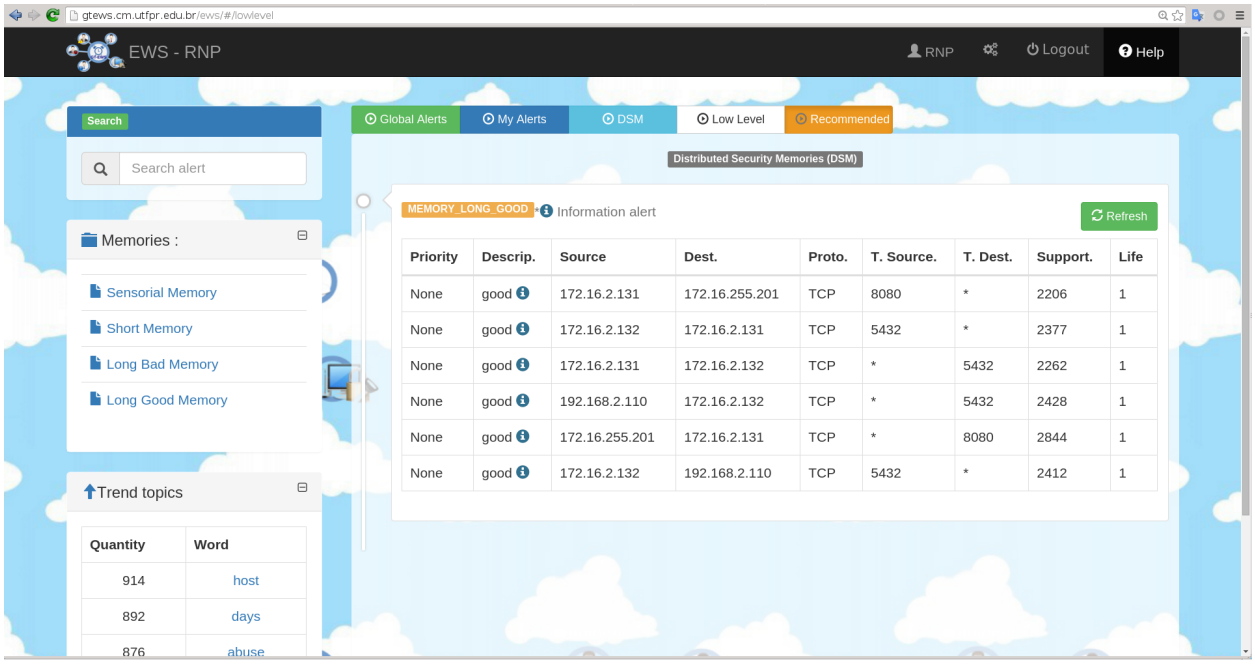


Figura 5.21: *Interface Web do GT-EWS apresentando o processamento da memória longa boa durante Experimento 4.*

Portanto, este experimento com o Of-IDPS em uma rede real em produção demonstra que o Of-IDPS é capaz de analisar e criar regras de segurança fundamentadas em um histórico de rede heterogêneo, ou seja, composto por vários alertas/ameaças. Também, a execução do Of-IDPS na rede não gerou perturbações indesejadas nos fluxos de redes considerados comuns e legítimos. Por fim, as memórias longa boa e ruim se mostraram efetivas em identificar o padrão de comportamento da LAN, pois criaram regras de segurança condizentes com o cenário de rede analisado.

5.1.2 Experimentos com fontes de alertas extraídas de redes sociais

Esta seção apresenta uma síntese dos alertas de segurança a respeito de cibersegurança que podem ser extraídos a partir de redes sociais, utilizando a metodologia proposta e implementada no submódulo Vulnerabilidades Externas do módulo Base de Conhecimento (ver Seção 4.2.5).

O objetivo desse experimento é verificar se há mensagens postadas em redes sociais que alertam a respeito de problemas de cibersegurança e se a metodologia proposta consegue evidenciar essas mensagens para que sejam utilizadas como fonte de informação auxiliando então na identificação de ameaças contra a segurança de ambientes computacionais.

Para este experimento foram coletados do Twitter, 155.631 *tweets* entre as datas de 28/04/2012 e 05/12/2012, ou seja, 222 dias. Tais *tweets* foram postados por 74.809 usuários do Twitter, sendo que aproximadamente 84,9% dos *tweets* continham URL, 37% possuíam *hashtags* (#) e 43% mencionavam (@) outros usuários no corpo da mensagem.

Todos *tweets* coletados foram submetidos ao passo "Filtrar Mensagens" (Figura 4.18), da arquitetura, que removeu cerca de 67% dos *tweets*. Pois, esses representavam mensagens mal formadas, fora do contexto de cibersegurança ou *spam*.

Os *tweets* restantes foram submetidos ao passo de "Agrupar por Similaridade", para desta forma evidenciar alertas de segurança. A relevância de cada grupo formado foi dada pela quantidade de usuários que propagaram a mensagem.

Uma amostra dos alertas de segurança extraídos do processamento desses *tweets* pode ser vista na Tabela 5.24, que apresenta três alertas de segurança para cada mês do experimento. A primeira coluna da tabela representa o mês em que os *tweets* foram coletados. A coluna *Msgs*, é a quantidade de mensagens similares que compõem o dado alerta. A coluna *Usu*, é o número de usuários que propagaram tal informação (usuários que redigiram uma mensagem no mesmo contexto ou sim-

plesmente replicaram (*retweet*)). A última coluna é o texto de um *tweet* do grupo que simboliza o alerta de segurança extraído. Então, por exemplo, a última mensagem de novembro ("Nov") relata que um novo *rootkit* pode enviar conteúdo malicioso para servidores HTTP Linux e tal informação foi postada 97 vezes, por 94 usuários do Twitter.

Tabela 5.24: *Os três alertas de segurança mais importantes de cada mês processado durante os meses do experimento.*

Mês	Msgs	Usu	Texto da mensagem
Mai	88	85	Microsoft boots Chinese firm for leaking Windows exploit..http://t.co/0QMqcQAl
	78	78	Android has made malware for Linux a reality. http://t.co/Qi5PxGcM
	72	70	...The Pirate Bay returns Anonymous hater takes credit for DDoS http://t.co/FQ0OrdTj
Jun	108	106	...Thousands of office printers hit by "gibberish"malware http://t.co/GgzSIFcx
	39	38	A virus specialized for AutoCAD a perfect cyber espionage tool... http://t.co/UhaHeXGn...
	29	29	...0day exploit taking advantage of an unpatched Windows / Office vuln is being exploited in the wild http://t.co/BovMm9Nd
Jul	151	144	Serial hacker says latest Android will be "pretty hard"to exploit...
	147	141	More malware found hosted in Google's official Android market...
	155	150	Security firm: Android malware pandemic by year?s end...
Ago	156	141	...30K workstations fell victim to cyber attack http://t.co/5dcvvKyS
	119	119	Attack against Microsoft scheme puts hundreds of crypto apps at risk...
	118	118	...MS-CHAPv2 puts hundreds of crypto apps at risk http://t.co/TQPsRkpN
Set	91	89	Emergency security patch issued by Microsoft to squash Internet Explorer zero day #Exploit... http://t.co/D1FXvkO7 @SecurityPhresh
	73	70	...New SSL/TLS attack for Hijacking HTTPS Sessions... http://t.co/J0IOy0Hj
	35	34	Attack Easily Cracks Oracle Database Passwords: Oracle?s software update for the flaw doesn?t protect http://t.co/o1ibSeuw #infosec
Out	208	194	Hacker Steals Millions of...Social Security Numbers...
	167	166	DSL modem hack used to infect millions with banking fraud malware...
	107	107	...Hacker Scores $60 000 From Google For Discovering Security Issue In Chrome...
Nov	133	133	...Mac OS X has more security holes than windows and they are easier to exploit too.
	98	95	New Linux rootkit injects mal HTML into Web servers...
	97	94	New Linux rookit injects malicious HTML into Web servers via ars technica...

Os alertas apresentados da Tabela 5.24 demonstram o potencial das redes sociais quanto a propagação de alertas. Por exemplo, há alertas que informam de maneira genérica a respeito de assuntos ligados cibersegurança, tal como os alertas:

- "*Android has made malware for Linux a reality...*" (segundo alerta do mês de maio);
- "*More malware found hosted in Google's official Android market...*" (segundo alerta do mês de julho).

Esse tipo de mensagem é conhecida nas redes sociais como *infosec*, já que são notícias a respeito de segurança de computadores. Na prática, analisando esses alertas o administrador da rede poderia rever as políticas de BYOD, já que aparentemente dispositivos móveis com Android possuem problemas com *malwares*. Nos aletas da Tabela 5.24, ainda existem outros alertas no mesmo contexto de notícia, que informam por exemplo a respeito de vazamento de dados (primeiro alerta de maio e outubro), ataques DDoS em sites de *torrent* (terceiro alerta de maio), estações de trabalho sendo afetadas por ciberataques (primeiro aleta de agosto), comparações de segurança entre sistemas operacionais (primeiro alerta de novembro).

Na Tabela 5.24, também há alertas de segurança mais contundentes e que poderiam afetar a rede ou seus recursos de forma mais direta. Alguns exemplos desses alertas são apresentados nos parágrafos a seguir.

Por exemplo, o primeiro alerta de junho ("*... Thousands of office printers hit by "gibberish"malware...*"), informa a repeito de um *malware* que afeta impressoras. Através desse alerta o administrador pode tomar medidas, tal como, a atualização do antivírus e regras de IDS para tentar impedir esse problema.

O segundo alerta de junho ("*A virus specialized for AutoCAD a perfect cyber espionage tool...*"), alerta a respeito de um vírus que envia projetos desenvolvidos no software AutoCAD para pessoas

não autorizadas (*hackers/crackers*), utilizando para isso serviços de e-mail do *host* da vítima. Assim, sabendo desse problema um administrador de rede pode implementar regras de *firewall*, *proxy* e/ou ferramentas anti-*spam* que impeçam o envio desses arquivos, ou simplesmente verificar se o antivírus consegue eliminar esse problema. Em casos mais radicais os administradores podem retirar o acesso à Internet dessas máquinas, já que esse vírus visa espionagem industrial.

Ainda no mês de junho, a última mensagem ("*...0day exploit taking advantage of an unpatched Windows / Office vuln is being exploited in the wild...*") informa a respeito de um *exploit* que utiliza-se de uma vulnerabilidade dia zero para afetar o Microsoft Windows e Office, ou seja, é um alerta de uma nova ameaça que pode nem estar presente em IDS e antivírus, na pior das hipóteses o administrador da rede pode ficar alerta sobre atividades anormais na rede.

A segunda ("*Attack against Microsoft scheme puts hundreds of crypto apps at risk...*") e a terceira (*...MS-CHAPv2 puts hundreds of crypto apps at risk...*) mensagens de agosto, informam a respeito de uma vulnerabilidade relacionada com o MS-CHAP (*Microsoft Challenge-Handshake Authentication Protocol*) versão 2. Caso o administrador constate o problema em sua rede, esse pode atualizar o software ou implementar regras no *firewall* impedindo que a vulnerabilidade seja explorada.

No mês de setembro, o primeiro alerta ("*Emergency security patch issued by Microsoft to squash Internet Explorer zero day #Exploit... http://t.co/D1FXvkO7 @SecurityPhresh*") é também a respeito de uma ameaça de dia zero ao navegador Internet Explorer da Microsoft, nesse caso o administrador pode, por exemplo, trocar ou atualizar o navegador dos computadores.

O segundo alerta de setembro ("*...New SSL/TLS attack for Hijacking HTTPS Sessions...*"), informa a respeito de ataques contra conexões HTTPS, provavelmente atualizações devem ser feitas para evitar esse tipo de ataque.

A última mensagem do mês de setembro ("*Attack Easily Cracks Oracle Database Passwords: Oracle?s software update for the flaw doesn?t protect...*") informa um problema com o sistema de autenticação de usuário e senha do banco de dados Oracle. Nesse o próprio alerta dá como solução a atualização do referido software.

O segundo alerta do mês de outubro ("*DSL modem hack used to infect millions with banking fraud malware...*"), informa que *malwares* infectaram dispositivos de redes para obter informações dos usuários das redes. Desta forma, o administrador da rede poderia verificar se os registros (*logs*) e configurações de seus dispositivos de rede, tal como *modems*, não foram alterados indevidamente.

No mês de outubro a segunda ("*New Linux rootkit injects mal HTML into Web servers...*") e terceira ("*New Linux Assimrookit injects malicious HTML into Web servers via ars technica...*") mensagens, noticiam a mesma informação, ou seja, que existem ferramentas que permitem explorar vulnerabilidades em servidores HTTP no sistema operacional Linux. Assim, o administrador de rede poderia verificar se não há nenhuma anormalidade na rede, referente a esse tipo de ação de *hackers*. Se for o caso, o administrador pode posteriormente realizar atualizações no servidor HTTP e no Linux, para eliminar o problema.

Ainda, para uma melhor apreciação dos resultados foi realizada uma análise manual e mais minuciosa nos alertas dos meses de junho e setembro (meses selecionados aleatoriamente). Tal análise teve como objetivo verificar manualmente, dentre as mensagens dos agrupamentos gerados pela metodologia (ver Seção 4.2.5), quais eram realmente alertas de segurança e quais não eram[13]. Assim sendo, concluiu-se que 65,7% das mensagens obtidas no mês de junho são alertas reais de cibersegurança. Já no mês de setembro os alertas totalizam 56,9% dos agrupamentos obtidos.

Portanto, analisando as mensagens obtidas para esse experimento verificou-se que os usuários do Twitter postam mensagens a respeito de cibersegurança. Também, analisando as mensagens/agrupamentos obtidos pela metodologia proposta, constata-se que é possível evidenciar alertas de cibersegurança a partir de mensagens postadas em redes sociais. Por fim, conclui-se que as redes sociais apresentam um bom potencial para serem utilizadas como sensores, ajudando assim a evidenciar problemas relacionados com cibersegurança (Campiolo *et al.*, 2013; Santos *et al.*, 2012, 2013).

[13] Representam notícias genéricas de cibersegurança, *spam* ou mensagem fora de contexto.

5.2 Resultados

Está seção apresenta sínteses e considerações a respeito dos resultados obtidos nos experimentos apresentados na seção anterior, bem como resultados de artigos e trabalhos oriundos das metodologias propostas nesta tese.

5.2.1 Resultados com fontes de alertas extraídas de redes sociais

Quanto ao uso de mensagens postadas em redes sociais como fonte de informação a respeito de ciberameaças. Os resultados obtidos durante o desenvolvimento da tese, deram origem inicialmente aos seguintes trabalhos:

- Em Santos *et al.* (2012) verificou-se que haviam informações a respeito de cibersegurança em mensagens postadas no Twitter e que tais mensagens apresentam ainda as seguintes características: são propagadas rapidamente, no geral são confiáveis e muitas informações são postadas primeiramente em redes sociais e posteriormente em sítios Web especializados;
- Em Campiolo *et al.* (2013), foram feitas melhorias ao método de agrupamento por similaridade proposto em Santos *et al.* (2012), o que diminuiu a aparição de mensagens de *spam* ou fora de contexto dentre as mensagens relacionadas com cibersegurança;
- No trabalho Santos *et al.* (2013) constatou-se que as mensagens postadas em redes sociais, no caso o Twitter, podem ser utilizadas como alertas de segurança contra ciberameaças. Para isso uma arquitetura fundamentada em (Campiolo *et al.*, 2013; Santos *et al.*, 2012) foi proposta, implementada e avaliada. Tal arquitetura é a mesma desta tese.

Em resumo, os resultados apresentados na Seção 5.1.2 e publicados em (Campiolo *et al.*, 2013; Santos *et al.*, 2012, 2013), demonstram que mais de 50% das mensagens obtidas, pela metologia proposta, são realmente alertas relacionados com cibersegurança. O que demonstra que as redes sociais apresentam potencial para servirem como fontes externas de informação a respeito de ciberataques, tal como foi proposto na tese. Motivado por esses resultados criou-se um grupo de trabalho/pesquisa que visa desenvolver um sistema de alertas antecipados a respeito de ciberameaças que utiliza, dentre outras, informações provindas de redes sociais. Tal grupo é apresentado a seguir.

5.2.1.1 GT-EWS

O Grupo de Trabalho *Early Warning System* é um projeto financiado pela RNP e tem por objetivo desenvolver um sistema de alertas antecipados a respeito de problemas relacionados com cibersegurança.

Tal projeto, até o momento, está em sua segunda fase, sendo que na primeira fase (com duração de 12 meses – novembro/2014 a outubro/2015), foi desenvolvida uma arquitetura e um protótipo de sistema de alertas antecipados, que pode fazer uso de fontes/sensores como: redes sociais; IDS; OpenFlow; etc, para gerar alertas de segurança. Após a primeira fase, a RNP entendeu que o projeto merecia mais tempo, para avançar no desenvolvimento de pesquisas referentes ao uso de informações escritas em português e que são postadas em redes sociais. Então, o projeto foi aprovado para uma segunda fase, com duração de 14 meses (novembro/2015 a dezembro/2016). A metodologia para obter e processar essas informações em português é parte da tese de Campiolo (2016).

Atualmente o GT-EWS está desenvolvendo o software Hórus, que é o nome dado ao sistema de alertas antecipado idealizado e desenvolvido durante o projeto.

Utilizando redes sociais como Twitter e Facebook, o Hórus consegue alertar a respeito de:

- Anúncios de vulnerabilidades;
- Orquestração de ataques;

- Vazamento de dados;
- Ataques de desfiguração de páginas Web;
- Ataques DoS/DDoS;
- Dentre outros.

A Figura 5.22 apresenta a interface do Hórus na primeira fase. Na parte central da interface é possível observar que o sistema está alertando, através de duas mensagens, a respeito de uma nova ameaça chamada Venom, que a principio afeta centros de processamentos de dados (*datacenters*). O primeiro alerta ("*Bigger than Heartbleed?! ?Venom? #zeroday vulnerability threatens most datacenters...*"), é composto por 794 *tweets*, postados por 749 usuários no período de 2 dias. Através do botão *Statistics*, da interface Web, o sistema mostra o montante de mensagens postadas durante esses dois dias (ver Figura 5.23). Já a Figura 5.24 apresenta os *tweets* que foram agrupados para formar o alerta gerado, essa tela é apresentada clicando-se no botão *Related tweets*.

Ainda na interface Web (ver Figura 5.22) é possível notar um menu lateral esquerdo, que permite:

- Buscar alertas por palavras (*Search alert*);
- Ver alertas por períodos de tempo (últimas 24 horas – *Recent*, últimos 3 dias – *Last Days*, última semana – *Week* e último mês – *Month*);
- Ver as palavras mais utilizadas nas mensagens de cibersegurança (*Trend topics*). No caso da Figura 5.22, a palavra em evidência é *venom*, com 2.971 ocorrências.

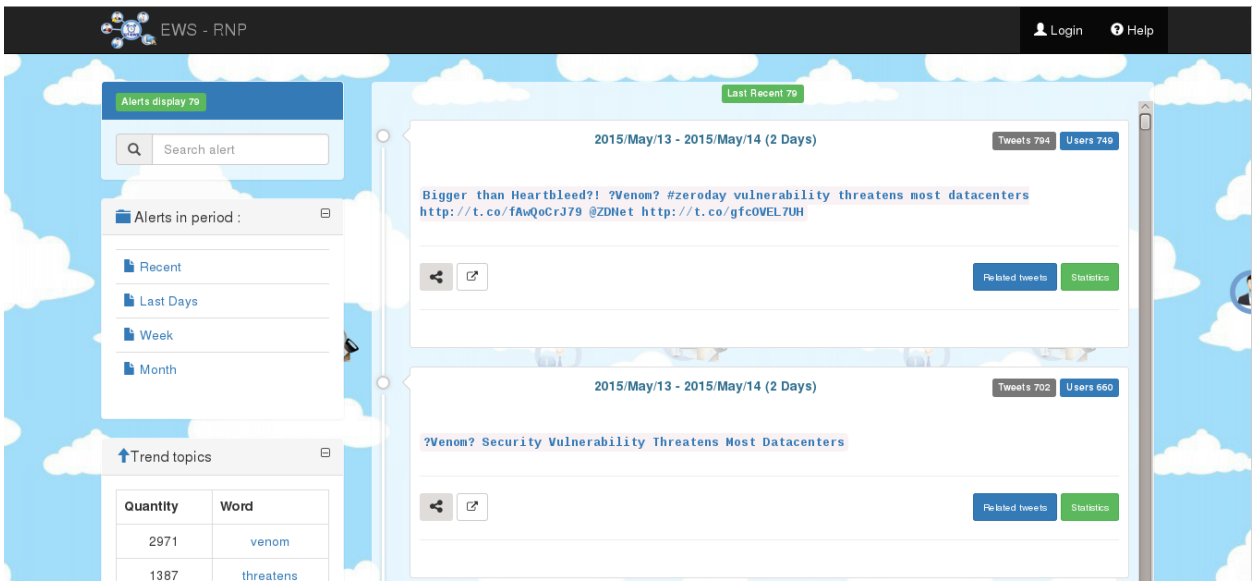


Figura 5.22: *Anúncio de vulnerabilidade na interface do Hórus.*

A Figura 5.25-a, apresenta um alerta a respeito de uma desfiguração de página em um sítio Web do governo brasileiro, tal sítio desfigurado pode ser visto na Figura 5.25-b. Atualmente o Hórus é capaz de capturar e armazenar tanto a postagem anunciando a desfiguração de páginas, quanto as imagens (*sreenshot*) dos sítios Web suspeitos de desfiguração. Essas capturas podem ser utilizadas posteriormente como provas de possíveis cibercrimes.

Para a segunda fase do GT-EWS, a interface Web foi reformulada e novas funcionalidades, tal como a identificação automática de entidades (atacante, alvo, etc) foram adicionadas, seguindo a metodologia proposta em (Campiolo, 2016). Essa nova interface é apresentada na Figura 5.26. Também, através dessa figura é possível observar duas mensagens alertando a respeito de um vazamento de dados, que são apresentadas na tela de últimos alertas (*Last Alerts!*). A nova interface

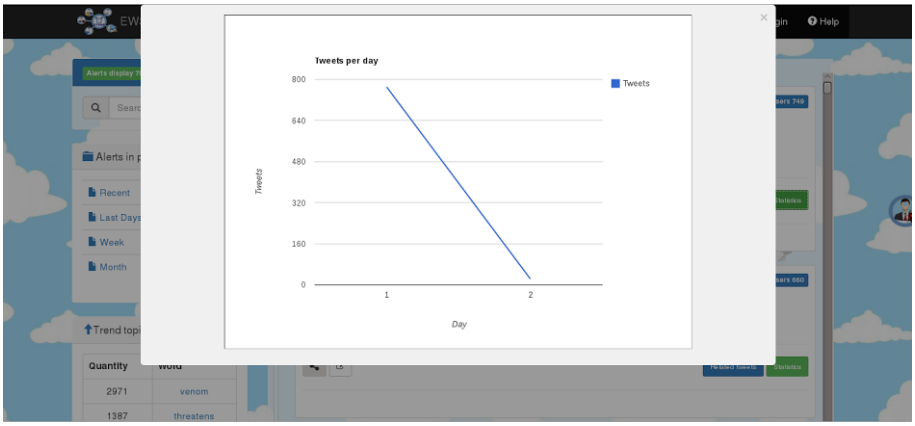


Figura 5.23: *Linha de tempo no Hórus.*

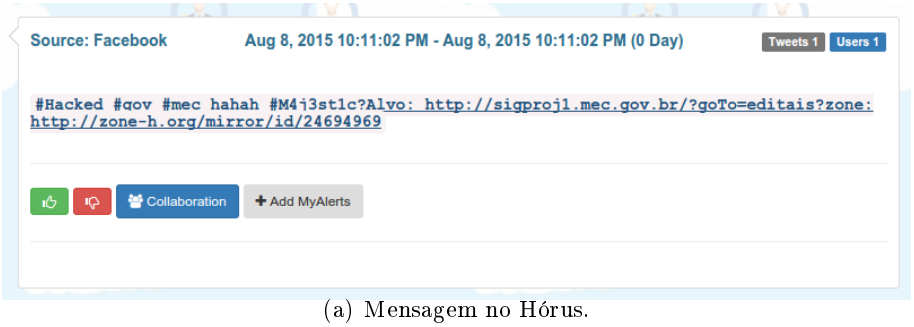
Figura 5.24: *Tweets que formam o alerta do Venom.*

também possui um *dashboard*, que dá acesso rápido a algumas informações do sistema (sensores, alertas, usuários, etc). O menu lateral esquerdo agora além de permitir o acesso aos alertas (*Alerts*), também dá acesso a informações de geolocalização das mensagens de redes sociais (*Geolocation*), à configurações/status dos sensores (*Sensors*) e demais configurações do sistema (*Configuration*), tal como configuração de usuários.

Um dos objetivos futuros do GT-EWS é correlacionar automaticamente alertas de alto nível (gerados a partir do processamento de mensagens postadas em redes sociais) com alertas de baixo nível (gerados a partir de fontes como IDS, *logs*, OpenFlow, etc). Tal funcionalidade permitirá gerar alertas mais contundentes se houverem atitudes suspeitas ocorrendo em redes locais, que ainda combinem com relatos de ameaças/vulnerabilidades postados em redes sociais. Contudo, criar um sistema para correlacionar alertas de baixo nível com os de alto nível não é uma tarefa trivial, pois por exemplo:

- As mensagens de redes sociais, que dão origem aos alertas de alto nível, são escritas de forma não estruturada, o que dificulta muito seu processamento e interpretação como alerta de segurança;
- Mesmo os alertas de baixo nível, tal como os do Snort NIDS, apresentam deficiências no que se refere a descrição e/ou detalhamento do problema de segurança que gerou o alerta (exemplo, falta de padronização), o que naturalmente dificulta a correlação automática com os alertas de alto nível.

Assim sendo, por enquanto a correlação entre alertas de baixo nível com os de alto nível deve ser feita manualmente pelo administrador da rede.



(a) Mensagem no Hórus.

(b) Página Web Desconfigurada.

Figura 5.25: *(a) Alerta informando uma desfiguração de página e (b) a página Web desfigurada.*

Atualmente, o principal usuário e interessado pelo sistema Hórus é o CAIS (Centro de Atendimento a Incidentes de Segurança) da RNP, que é responsável por detectar e mitigar ciberataques que afetam instituições públicas Brasileiras (dai o interesse por mensagens escritas em português), tal como ataques contra universidades. Além do CAIS, o GT-EWS têm como parceiros a UFBA (Universidade Federal da Bahia), PRODAM (Processamento de Dados Amazonas S/A) e a Polícia Federal. Informações a respeito do GT-EWS estão disponíveis em https://gtews.ime.usp.br/.

Sustentados pelos resultados obtidos e pelas funcionalidades desejadas para o Hórus, mais dois trabalhos foram publicados. Sendo o primeiro Batista *et al.* (2016), que apresenta a arquitetura empregada no Hórus para a extração de alertas de cibersegurança e o segundo Esposte *et al.* (2016), que propõem uma funcionalidade nova/futura, que é a recomendação de alertas fundamentada nos perfis dos administradores de rede que utilizam o Hórus, tal como sugere a tese de Campiolo (2016).

Por fim, a metologia proposta pelo GT-EWS dá aos administradores de rede uma nova e rica fonte de informações a respeito de cibersegurança, que são os alertas de alto nível, que podem auxiliar administradores de rede a tomar medias de segurança mais assertivas para manter as redes seguras.

5.2.2 Resultados do Of-IDPS

A seguir são apresentados os resultados referentes a metologia autonômica proposta para mitigar ataques contra a segurança de redes locais.

Parte dos resultados da tese são apresentados nos artigos Santos *et al.* (2014) e Santos *et al.* (2016). Sendo que, em Santos *et al.* (2014) foi proposto uma arquitetura autonômica que combina tecnologias legadas (IDS), com novas tecnologias (RDS), para criar sensores e atuadores, bem como uma base de conhecimento a respeito do uso da rede. Assim, o resultado foi a criação do Of-IDPS, que permite perceber o que está ocorrendo em uma rede local, registrar essas informações e reagir frente as possíveis ameaças, utilizando atuadores.

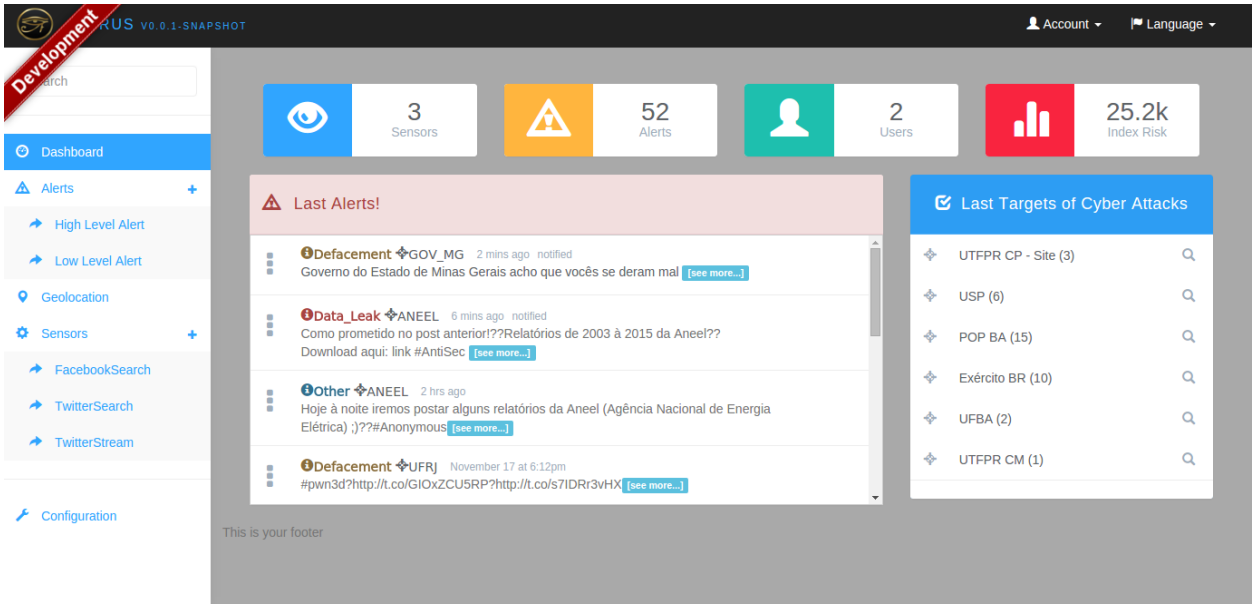


Figura 5.26: *Interface atual do Hórus.*

Já usando como base a arquitetura desenvolvida em Santos *et al.* (2014), o trabalho Santos *et al.* (2016) apresentada a metodologia que permite analisar os históricos da rede e caso necessário, planejar ações que visem manter ou restabelecer o bom andamento da rede. Então, o resultado é o desenvolvimento de métodos que empregam técnicas de aprendizado de máquina não supervisionada, que processam as informações da base de conhecimento da rede, de forma similar a estrutura da memória humana, para então gerar autonomicamente regras de segurança que têm por objetivo manter em funcionamento fluxos de redes legítimos e mitigar fluxos maliciosos.

5.2.2.1 Resultados dos experimentos com DoS

Durante os experimentos com ataques DoS, a rede com Of-IDPS conseguiu resultados melhores em todos os cenários de testes, quando comparada com a rede sem Of-IDPS.

A Tabela 5.25, apresenta os resultados compilados dos experimentos com DoS. Nessa tabela a coluna "Experimentos", indica o experimento. A coluna "Uso da Memória", informa quais memórias do Of-IDPS foram empregadas no experimento. A coluna "Bytes Enviados/Recebidos", mostra uma comparação percentual, no que se refere a envio/recebimento de *bytes* do servidor (fluxo legítimo) e vítima (fluxo malicioso), da rede com Of-IDPS em relação a rede sem Of-IDPS. Essa análise utiliza como referencia apenas o envio de *bytes*, pois os ataques eram de inundação. Não são apresentados os dados do atacante, pois o objetivo do Of-IDPS é proteger a vítima. Também, não são apresentados os dados do cliente, pois entende-se que o servidor só fornece uma resposta caso haja comunicação com o cliente. Assim, para simplificar a análise dos resultados os dados referentes ao atacante e cliente são omitidos, bem como demais dados (número de pacotes, etc) obtidos durante os experimentos, mas esses estão disponíveis na Seção 5.1.1.

Desta forma, analisando a Tabela 5.25 constata-se que a vítima da rede com Of-IDPS na média obteve uma redução de 76% na quantidade de *bytes* gerados pelo ataque DoS, quando comparada com a vítima da rede sem Of-IDPS. O que demonstra a efetividade da solução autonômica proposta em mitigar ataques DoS.

Também nota-se que a reação autonômica gerada para mitigar o ataque DoS, conseguiu atenuar os efeitos do ataque nos fluxos legítimos, pois na maioria dos casos houve uma melhora/aumento na quantidade de *bytes* enviados/recebidos no servidor com Of-IDPS se comparado ao servidor sem Of-IDPS (ver Tabela 5.25). O único caso que não houve aumento percentual foi no experimento 2.2, já que nesse os fluxos maliciosos e legítimos são destinados a *hosts*/serviços de redes distintos. Ou seja, neste cenário o ataque não afeta os fluxos legítimos e devido a isso não houve diferenças

Tabela 5.25: *Compilação dos resultados das diferenças percentuais dos bytes enviados e recebidos por servidor e vítima nos experimentos com DoS.*

Experimento	Uso da Memória				*Bytes* Enviados/Recebidos (%)	
	Sensorial	Curta	Longa Boa	Longa Ruim	Servidor	Vítima
2.2	✓	✓			■ 0,00	▼ 68,49
2.3	✓	✓			▲ 446,00	▼ 72,90
2.4	✓	✓			▲ 44,95	▼ 71,01
2.8	✓	✓	✓		▲ 588,90	▼ 71,82
2.9	✓	✓	✓	✓	▲ 769,74	▼ 99,99

▲ Aumento percentual.
▼ Redução percentual.
■ Não apresentou diferença percentual.

nos cenários com e sem Of-IDPS.

Ainda, observando-se os dados do Servidor na Tabela 5.25, também nota-se que a reação autonômica gerada não afetou a transmissão de *bytes* dos fluxos legítimos. Exceto, no Experimento 2.4, no qual fluxos maliciosos e legítimos foram destinados ao mesmo *host* e serviço de rede. Inicialmente, nesse experimento o fluxo legítimo sofreu interferência negativa da reação autonômica, contudo em um segundo momento as regras autonômicas convergiram para mitigar apenas o fluxos malicioso e o fluxo legítimo foi normalizado. Entretanto, esse problema[14] é sobrepujado nos Experimentos 2.8 e 2.9. Para tanto utiliza-se a memória longa, que analisa o histórico da rede para identificar os fluxos bons e ruins da rede. O que demonstra a efetividade da estrutura de memória empregada no Of-IDPS.

5.2.2.2 Resultados dos experimentos com DDoS

Nos experimentos com ataques DDoS, o Of-IDPS também conseguiu proteger a rede e reduzir em todos os casos o fluxo de dados do atacante para a vítima.

Conforme pode ser visto na Tabela 5.26 (os campos dessa tabela são similares ao da Tabela 5.25), em todos os experimentos a vítima apresentou uma redução de no mínimo 67,98% na quantidade de *bytes* enviados/recebidos em relação a vítima da rede sem Of-IDPS. Ou seja, a metologia empregada no Of-IDPS mostrou-se capaz de mitigar ataques de DoS mais complexos, como é o caso dos ataques DDoS com endereços IPs de origem desconhecida e aleatória, que foram executados nesses experimentos.

Tabela 5.26: *Compilação dos resultados das diferenças percentuais dos bytes enviados e recebidos por servidor e vítima nos experimentos com DDoS.*

Experimento	Uso da Memória			*Bytes* Enviados/Recebidos (%)	
	Sensorial	Curta	Longa Boa	Servidor	Vítima
2.5	✓	✓		▼ 79,94	▼ 68,33
2.6	✓	✓		▲ 541,33	▼ 69,11
2.7	✓	✓	✓	▲ 579,40	▼ 67,98

▲ Aumento percentual.
▼ Redução percentual.

Quanto a influência do ataque DDoS e das regras do Of-IDPS, sob os fluxos legítimos, os resultados são apresentados nos parágrafos seguintes.

No Experimento 2.5, que utiliza apenas as memórias sensorial e curta em um cenário em que tanto fluxos maliciosos quanto legítimos são destinados a um mesmo *host*/serviço de rede. Observa-se, que o Of-IDPS bloqueou também os fluxos legítimos destinados ao *host*/serviço sob ataque, na tentativa de conter os fluxos maliciosos gerados durante o DDoS. Devido a isso o servidor apresentou

[14] Problema da reação autonômica interferir no fluxo legítimo, quando ambos fluxos (legítimo e malicioso) são destinados ao mesmo *host*/serviço de rede.

uma redução de 79,94% na transmissão de *bytes* (ver Tabela 5.26). Isso ocorre porque a origem do ataque é desconhecida e aleatória. Então, para impedir que a rede e/ou seus *hosts*, sofram com mais interferências do ataque, a metodologia da memória curta do Of-IDPS gera regras de segurança que mitigam todos os fluxos destinados ao serviço de rede que está sob ataque naquele *host*.

Todavia, adicionando a memória longa boa no Of-IDPS, esse se torna capaz de distinguir os fluxos legítimos dos maliciosos. Ou seja, através da memória longa boa o Of-IDPS é capaz de evitar que os fluxos legítimos sofram interferências de um ataque DDoS, mesmo que o fluxo legítimo seja destinado ao mesmo *host*/serviço que está sob ataque. É justamente isso que demonstra o Experimento 2.7 (ver Tabela 5.26), pois o montante de *bytes* transmitidos pelo servidor aumentou em 579,40% em relação a rede sem Of-IDPS. O que demonstra que a análise do histórico da rede ajuda o Of-IDPS a mitigar fluxos de redes considerados maléficos e manter em funcionamento fluxos considerados normais.

Já no Experimento 2.6, no qual fluxos legítimos e maliciosos são transmitidos para serviços de redes diferentes de um mesmo *host*. Observa-se, que as regras autonômicas conseguem mitigar o ataque e restaurar a transferência de dados dos fluxos legítimos (ver Tabela 5.26). Ou seja, durante um ataque DDoS de fonte desconhecida, o Of-IDPS utilizando apenas as memórias sensorial e curta, consegue mitigar o ataque para um dado serviço e manter em funcionamento os demais serviços desse mesmo *host*.

5.2.2.2.1 Efeitos dos ataques DoS e DDoS notados na rede OpenFlow

Durante os experimentos com DoS e DDoS (Seção 5.1.1.2), percebeu-se que esse tipo de ataque pode causar efeitos colaterais nos equipamentos OpenFlow.

Isso ocorre porque, dependendo da configuração, é exigido que o controlador sempre seja questionado a respeito de novos fluxos que chegam em *switches* (ver Seção 2.3.1). O que pode ocasionar um excesso de mensagens de controle (*overhead*) na rede e tais mensagens, podem intencionalmente ou não, comprometer o bom funcionamento da rede.

Por exemplo, durante uma transmissão de rede TCP convencional, sem tráfego malicioso, que envia pouco mais de 14.000 pacotes, são transmitidos 34 pacotes OpenFlow entre o controlador e o *switch*. Tais pacotes OpenFlow de forma resumida representam as seguintes mensagens:

1. Pedido (*in*) partindo do *switch* para que o controlador encontre a porta de destino, para estabelecer fluxo do cliente para o servidor;

2. Comando do controlador para adicionar (*add*) o fluxo no *switch* e permitir transmissão de dados do cliente para o servidor;

3. Pedido (*in*) partindo do *switch* para que o controlador encontre a porta de destino, para estabelecer fluxo do servidor para o cliente;

4. Adição (*add*) do fluxo servidor para o cliente no *switch*.

Dependendo o caso, também serão necessárias mensagens OpenFlow para tratar fluxos ARP (*in/out/in/add*) e DNS (*in/add/in/add*), relacionados com a transmissão TCP. O que demonstra que as redes OpenFlow exigem naturalmente mais mensagens de controle se comparadas com uma rede tradicional sem OpenFlow.

Logo, o envio/recebimento de mensagens de controle entre os dispositivos OpenFlow, pode ser potencializado durante ataques DoS e DDoS. Por exemplo, a Figura 5.27 apresenta o trânsito de pacotes de controle OpenFlow enviados/recebidos entre controlador e *switch*, durante um ataque DoS TCP-SYN que envia 10.000 pacotes, em cenários de rede sem Of-IDPS (linha vermelha marcada com um círculo) e com Of-IDPS (linha azul marcada com uma estrela). Nessa figura observa-se que a taxa de pacotes de controle OpenFlow na rede sem Of-IDPS permanece em pico durante todo ataque, enquanto a rede com Of-IDPS apresenta um pico inicial e depois a taxa de pacotes OpenFlow cai em relação a rede sem Of-IDPS. Isso ocorre porque o atacante altera a todo momento a porta

de origem do ataque, o que exige que o *switch* envie pacotes OpenFlow para o controlador, que deve decidir se tais fluxos serão ou não instalados. Desta forma, na rede sem Of-IDPS o controlador recebe esse pedido e defere a instalação do referente fluxo nos *switches*, o que acarreta em mais mensagens de controle. Já na rede com Of-IDPS, tal pedido de instalação de fluxos maliciosos são indeferidos pelo controlador, assim que o ataque é identificado, o que consequente reduz a quantidade de pacotes OpenFlow que trafega entre o controlador e os *switches*.

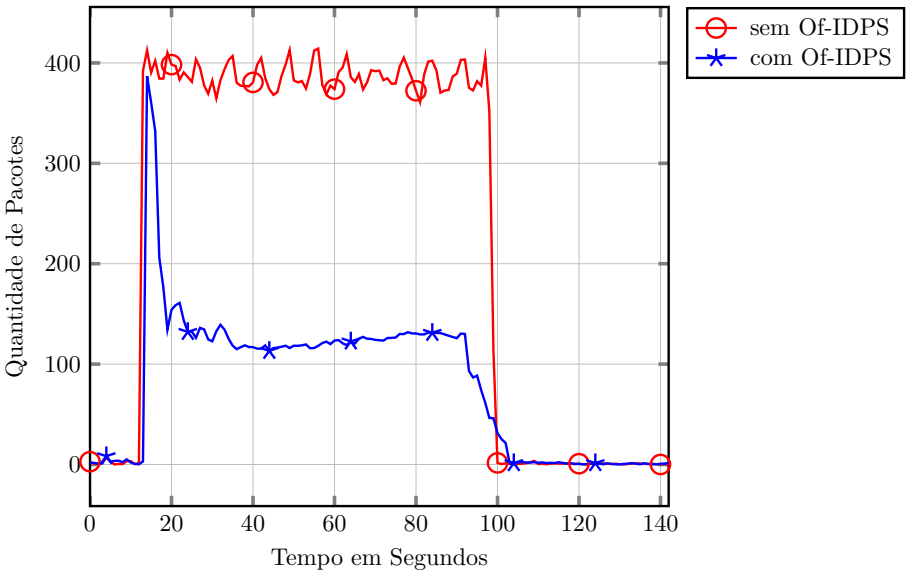


Figura 5.27: *Fluxo de pacotes OpenFlow no controlador durante ataque de inundação TCP-SYN*

A Figura 5.28 ilustra apenas o envio/recebimento de mensagens de controle OpenFlow, que representam a adição de novos fluxos, do exemplo anterior. Dessa maneira, observando a Figura 5.28, constata-se que na rede OpenFlow sem nenhuma proteção e sob ataque DoS, há um trânsito constante de mensagens de controle com pedidos de adição de novos fluxos entre o controlador e os *switches*. Já no cenário de rede que utiliza o Of-IDPS (linha azul marcada com estrela), os pacotes de adição de fluxos maliciosos só ocorrem no início, quando o ataque ainda não havia sido confirmado, depois não há mais adição de fluxos para os pacotes do ataque DoS, o que mitiga os efeitos do ataque na rede.

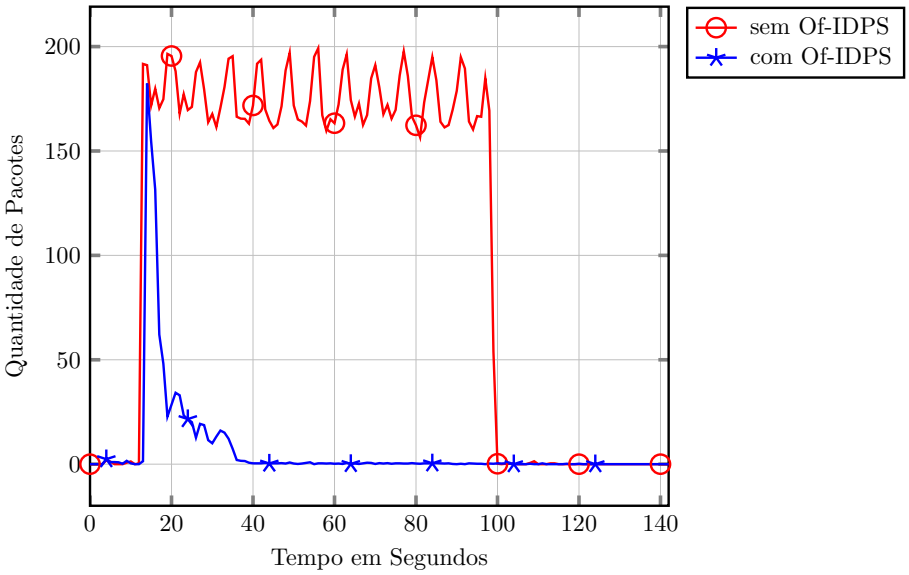


Figura 5.28: *Fluxo de pacotes OpenFlow para adicionar fluxos em switches durante o ataque DoS*

No exemplo apresentado anteriormente (Figuras 5.27 e 5.28), no cenário de rede sem Of-IDPS, no qual utilizou-se OpenFlow para executar o algoritmo de comutação tradicional, o controlador sob ataque DoS recebeu em média 33.457 pacotes, enquanto no cenário de rede com Of-IDPS o controlador recebeu 11.435 pacotes. Ou seja, a rede com o Of-IDPS tratou apenas 34% dos pacotes OpenFlow referentes a fluxos maliciosos. E o mais importante o controlador com o Of-IDPS enviou apenas 935 pacotes de controle OpenFlow para que os *switches* instalassem fluxos para os pacotes do ataque DoS, contra 15.177 pacotes da rede sem o Of-IDPS. Assim sendo, o ambiente que utilizou a arquitetura autonômica proposta instalou apenas 6% dos fluxos maliciosos nos *switches*, o que economiza recursos e evita o efeito colateral do ataque DoS no controlador OpenFlow e *switches*. Shin *et al.* (2013b) também identificou o problema e aponta uma solução que envolve a modificação no controlador para lidar com essas questões em aplicações de segurança.

Enfim, essa sobrecarga de mensagens de controle OpenFlow depende de alguns aspectos para ocorrer, tal como da programação e configuração dada à rede OpenFlow, mas em linhas gerais é um problema em aberto dentro das pesquisas que envolvem RDS (Jesus *et al.*, 2014).

5.2.2.3 Outros resultados

Durante os Experimentos 3 e 4, o Of-IDPS foi submetido a vários tipos de ameaças, inclusive foi posto a prova em um ambiente de rede real em produção.

No Experimento 3 (Seção 5.1.1.3), foram injetados em uma rede simulada vários pacotes com assinaturas de ataques normalmente conhecidos por IDS. Nesse cenário o Of-IDPS, interagindo com o Snort NIDS, detectou tais pacotes, analisou-os e gerou regras de segurança que bloquearam 55% desses pacotes e retardaram o tempo do ataque em até 276%, quando comparado com a rede sem Of-IDPS. O que demonstra que o Of-IDPS consegue utilizar sistemas de segurança legados como sensores, no caso o Snort. Também os resultados mostram que a metodologia do Of-IDPS consegue reagir a vários tipos de ameaças.

No Experimento 4 (Seção 5.1.1.4), o Of-IDPS foi instalado e configurado para manter a segurança de uma rede real em produção. Durante os 6 meses em que o Of-IDPS monitorou a rede, foram emitidos 15.098 alertas de segurança, que representaram 44 tipos de ciberameaças. Contudo em nenhum momento constatou-se que a segurança dessa rede foi comprometida. O que demonstra a efetividade do Of-IDPS em mitigar ataques em uma rede real em produção. Outro resultado importante obtido nesse experimento, foi a constatação prática de que a defesa autonômica não afeta negativamente os fluxos de rede legítimos, que circulam na LAN.

Um resultado interessante, obtido da execução do Of-IDPS na rede real em produção (Experimento 4), é que o Of-IDPS permite que o administrador da rede visualize de forma resumida quais são os fluxos mais frequentes em sua rede, bem como os principais ataques sofridos. Tal visualização se torna factível ao agregar-se uma interface Web que apresente as regras de segurança contidas em cada memória do Of-IDPS. Essa funcionalidade mostra-se útil, pois muitas vezes o administrador tem que analisar centenas ou milhares de alertas e fluxos de rede para saber qual é o comportamento real da rede. Por exemplo, para uma pessoa sintetizar o uso e as ameaças da rede do Experimento 4, seria necessário analisar mais de 15 mil alertas de segurança e 10 milhões de fluxos OpenFlow. Todavia muitas vezes, o administrador não tem tempo ou paciência para fazer isso. Porém, utilizando o Of-IDPS e a interface Web é possível obter essa informação sintetizada em algumas linhas (ver Figuras 5.20 e 5.21). O que pode ser de grande valia na luta do administrador em manter a rede e seus ativos seguros e em pleno funcionamento.

Analisando os sensores empregados no Of-IDPS, observa-se que as tabelas de fluxos OpenFlow proveem informações que possibilitam e/ou auxiliam na confirmação de determinados tipos de ciberameaças. Por exemplo, os dados obtidos dos *switches* OpenFlow permitiram identificar o uso anormal da rede durante os ataques DoS e DDoS. Pois, observou-se o aumento incomum no número de requisições para a instalação de fluxos OpenFlow e o baixo número de pacotes transmitidos por cada fluxo relacionado com esses ataques, o que ajudou a caracterizar o comportamento suspeito. Ou seja, as tabelas de fluxos OpenFlow mostraram-se bons sensores, pois são fonte de informação a respeito do uso da rede.

Já quanto ao uso de sensores heterogêneos (IDS e OpenFlow), isso também mostrou-se uma boa estratégia. Pois, as informações de um sensor podem complementar as de outro, fornecendo ao Of-IDPS diferentes visões/dados a respeito do uso da rede e de ataques que estejam ocorrendo. Desta forma, a correlação dessas informações pode elucidar dúvidas a respeito de possíveis ataques na rede.

Em relação ao uso da tecnologia OpenFlow como atuador em uma arquitetura autonômica. Essa se mostrou muito versátil, pois permite que o Of-IDPS reprograme a rede de várias maneiras (Experimento 1), utilizando apenas uma única interface de interação (mensagens OpenFlow). O que ajuda a coordenar toda a rede durante a mitigação de fluxos de redes maliciosos, sem comprometer a operação dos demais fluxos da rede.

Durante a revisão bibliográfica constatou-se que não há muitos trabalhos que tratam especificamente do ato da reação autonômica para mitigar problemas de segurança em LANs. Principalmente combinando CA, RDS e métodos de aprendizagem de máquina não supervisionado, para combater fluxos de rede maliciosos e zelar pelos fluxos legítimos. Muito menos trabalhos que tentem utilizar informações de redes sociais para manter o administrador da rede informado a respeito de vulnerabilidades que estão surgindo.

Assim sendo, o trabalhos apresentados em Chen *et al.* (2014) e Chu *et al.* (2010) são os mais similares com o presente trabalho, mas apenas por dar enfase na reação autonômica em redes de computadores. A solução apresentada em Chu *et al.* (2010) consegue reduzir em 65,72% a quantidade de pacotes enviados do atacante para a vítima durante um ataque DDoS. Já o Of-IDPS consegue reduzir em 95,1%, a quantidade de pacotes enviados do atacante para a vítima (ver Tabela 5.2), em ataques DDoS. Em Chen *et al.* (2014), os autores apresentam gráficos referentes a mitigação de fluxos maliciosos durante ataques DoS, mas infelizmente não relatam os valores referentes a quantidade de pacotes ou *bytes* enviados/recebidos, o que dificulta a comparação entre os trabalhos.

Outro ponto de comparação direta entre os trabalhos, em termos de resultados obtidos, é o tempo da reação autonômica para mitigar ciberataques. Sendo que a proposta de Chu *et al.* (2010) consegue reagir à ataques DDoS e iniciar o bloqueio dos pacotes maliciosos em 10 segundos. Nos experimentos com DoS em Chen *et al.* (2014), os autores conseguiram mitigar os fluxos maliciosos depois de 95 segundos, do inicio do ataque. Já utilizando o Of-IDPS, a mitigação dos ataques DoS e DDoS de inundação TCP, dos Experimentos 2.2, 2.3, 2.4, 2.5, 2.6, 2.7 e 2.8, ocorreram em aproximadamente 45 segundos depois do inicio do ataque. Todavia, no Experimento 2.1 o Of-IDPS conseguiu mitigar ataques DDoS TCP-SYN em 10 segundos. Sendo ainda que o Of-IDPS apresenta um tempo de resposta de aproximadamente 7 segundos, do inicio do ataque até alguma reação do sistema (ver Experimento 1). Relembrando que os 45 segundos obtidos no Of-IDPS (mencionados anteriormente), são consequência dos pacotes maliciosos esperando para serem entregues nos *switches* OpenFlow, antes da reação autonômica coibir tais pacotes. Assim, esse tempo pode ser reduzido utilizando-se *switches* implementados em hardware, já que esses normalmente apresentam desempenho melhor do que *switches* implementados em software, como é o caso do Open vSwitch utilizado nos experimentos (Costa *et al.*, 2016). De qualquer modo, tanto o resultado obtido aqui, quanto em Chen *et al.* (2014) e Chu *et al.* (2010) podem ser considerados satisfatórios se comparados ao tempo que um administrador humano levaria para identificar e mitigar os problemas. É valido observar que ataques que ocorram em menos de 5 segundos (tempo de detecção no Of-IDPS – ver Seção 5.1.1.1) podem burlar as contramedidas criadas pelo Of-IDPS utilizando as memórias sensorial e curta, contudo serão mitigados pela memória longa ruim, caso persistam.

Por fim, os resultados desta seção demonstram que é possível desenvolver uma arquitetura autonômica que utiliza-se de sensores tal como OpenFlow e o Snort NIDS para analisar o que está ocorrendo atualmente na rede e caso existam problemas de segurança, empregar técnicas de aprendizagem de máquina para criar contramedidas, na forma de regras de segurança, que são aplicadas na rede por atuadores, tal como controladores e *switches* OpenFlow, para então mitigar ameaças e restaurar o equilíbrio da rede (Santos *et al.*, 2014). Constatou-se também, que através da metodologia proposta é possível analisar o histórico da rede para identificar fluxos comuns e

maliciosos. Depois, fundamentado nessa análise, moldar o comportamento da rede para evitar que fluxos comuns sejam afetados pela própria defesa autonômica e principalmente para prevenir que ataques afetem a rede no futuro (Santos *et al.*, 2016).

Capítulo 6

Conclusões

Neste capítulo são apresentados as considerações finais do trabalho e sugestões para pesquisas futuras.

6.1 Considerações finais

A arquitetura autonômica proposta por esta tese consegue identificar desequilíbrios causados por problemas de segurança e reagir, evitando a degradação massiva dos recursos da rede, *hosts* e/ou serviços de rede. Tudo isso com o mínimo de interação humana possível, pois a metodologia proposta emprega técnicas de aprendizagem de máquina não supervisionada para processar, similarmente a memória humana, o histórico de uso da rede e de alertas de segurança, para então criar/manter regras de segurança que protejam os fluxos de rede legítimos, dos fluxos de rede relacionados com ciberataques.

Dentro do cenário autonômico proposto pela tese, a integração de sistemas legados com tecnologias de vanguarda, tal como Snort e OpenFlow se mostrou muito eficaz, pois o uso dessas como sensores dão um panorama geral do estado da rede para a arquitetura proposta. Já quanto ao uso do OpenFlow como atuador, esse se mostrou uma facilitador no que se refere a interação com os elementos de rede e também permitiu reagir rapidamente (aproximadamente 7 segundos) frente aos problemas identificados na rede.

Quanto a metodologia empregada nas memórias sensorial e curta da arquitetura. Nos experimentos em redes simuladas, constatou-se que as regras de segurança criadas por essas memórias são capazes de mitigar problemas pontuais na rede (memória sensorial) e também aprender com as características do ataque para mitigar o problema de forma mais ampla (memória curta), evitando assim que *hosts* ou até mesmo toda a rede seja comprometida por um dado ataque. Tais memórias conseguiram, por exemplo, mitigar ataques DDoS evitando que até 97,5% dos pacotes maliciosos chegassem à vítima.

O uso das memórias longas dão a arquitetura autonômica proposta, a possibilidade de distinguir quais são os principais ataques que afetam a rede e também quais são os serviços de rede mais utilizados. Inclusive essas informações podem ser vistas pelo administrador da rede via interface gráfica, o que pode ser útil em tomadas de decisões futuras. Com o uso da memória longa ruim, nos experimentos com rede simulada, foi possível mitigar até 99,95% dos pacotes maliciosos enviados durante ataques DoS e simultaneamente, a metodologia da memória longa boa, conseguiu proteger os fluxos de rede que não estavam relacionados com os ataques, aumentando assim a quantidade de pacotes transmitidos nos fluxos legítimos em 752,42%. Além disso, instalando a arquitetura proposta em uma rede real em produção, constatou-se que a metodologia das memórias consegue na prática, proteger uma LAN de ciberataques sem comprometer os demais fluxos de rede (fluxos não relacionados com problemas de segurança). Pois, durante seis meses de testes a arquitetura protegeu a referente rede de 44 tipos de ciberameaças, sem causar distúrbios na rede.

Por fim, a metodologia proposta para extrair alertas de cibersegurança de fontes abertas e não estruturadas, tal como de redes sociais, se mostrou plausível. Pois, durante os experimentos

constatou-se que mais de 50% das mensagens extraídas são realmente alertas de cibersegurança. Assim, por enquanto, esses alertas ficam disponíveis ao administrador da rede de forma simplificada (em uma interface gráfica com mensagens agrupadas por similaridade) e no mínimo servem de complemento a outras fontes de informações (alertas de IDS, *logs*, dentre outras). Podendo dessa maneira ajudar em decisões futuras a respeito de providências para manter-se a segurança das redes. Mas futuramente espera-se que esses alertas extraídos de fontes não tradicionais sejam utilizados para identificar efetivamente e de maneira automática ciberameaças dentro das LANs.

6.2 Sugestões para pesquisas futuras

Devido as várias partes que podem compor a arquitetura autonômica proposta, a presente tese dá a possibilidade de pesquisas, trabalhos e/ou expansões futuras nos temas apresentados nos parágrafos seguintes.

Pesquisa, implementação e integração de novos sensores com a arquitetura autonômica proposta. Nesse sentido novos tipos de sensores podem trabalhar de forma cooperativa aos sensores OpenFlow e ao Snort, já implementados. Por exemplo: (i) Adicionando-se NIDS baseados em comportamento aumenta-se as chances da arquitetura identificar e reagir contra ataques desconhecidos ou novos; (ii) Incluindo HIDS como sensores da arquitetura, torna-se possível saber o estado atual dos *hosts* da rede, o que pode melhorar o nível de detecção dos ataques; (iii) Também, é pertinente a criação/implementação de mais métodos que expandam a capacidade dos sensores OpenFlow em identificar ciberameaças.

Além da adição de novos sensores, também é interessante que a arquitetura autonômica proposta possa ter outras opções de atuadores. Tal como, atuadores na forma de *firewalls*, *proxys*, roteadores ou até mesmo um software agente que pode ser instalado em *hosts* estratégicos da rede (por exemplo, servidores), permitindo assim que a arquitetura autonômica proposta tenha ainda mais controle sobre os elementos da rede. Esses novos atuadores poderiam substituir e/ou trabalhar de forma cooperativa com os atuadores OpenFlow, já existentes. Em conjunto com as pesquisa a respeito de novos atuadores, também é conveniente analisar outras formas de reações para combater suspeitas de ataques, que não só o bloqueio de pacotes de rede ou a redução de largura de banda.

Outro ponto que pode ser explorado por pesquisas futuras, são os métodos empregados no módulo Análise e Planejamento, da arquitetura autonômica proposta. Assim, novos métodos podem substituir ou apoiar os métodos já criados, na tarefa de analisar as informações da rede, para então criar planos de ações para manter ou restabelecer o equilíbrio da rede.

Pesquisar a viabilidade de expandir a arquitetura autonômica proposta de centralizada/local para distribuída/global. Sendo que, uma das possibilidades é ter um sistema federado, composto por várias LANs que compartilham os alertas de segurança e os dados de uso da rede gerados pela arquitetura. Desta maneira, esses dados compartilhados poderiam ser analisados pela metodologia das memórias da arquitetura proposta aqui, para então gerar regras de segurança em um âmbito mais global (fundamentado em dados de várias redes), além do âmbito local já proposto por esta tese.

A extração de alertas de cibersegurança a partir de mensagens postadas em redes sociais dão a possibilidade de pelo menos mais três temas de pesquisas, sendo esses: (i) Melhorar a metodologia para reduzir a quantidade de falsos positivos, aumento assim a precisão na extração de alertas a respeito de ciberameaças; (ii) Exploração e agregação de outras fontes de dados não estruturados a metodologia proposta, que não só o Twitter; (iii) Correlacionar os alertas de baixo nível[1] com os alertas de alto nível[2], para desta forma colaborar com a evolução dos sistemas de detecção de intrusão, que passariam a ter mais uma importante fonte de dados, que são as mensagens postadas por pessoas em redes sociais ou em outras formas de mídia disponíveis na Internet.

[1] Alertas de IDS, *logs*, dentre outros
[2] Alertas gerados de fontes como redes sociais

Apêndices

Apêndice A

Outros trabalhos relacionados

O texto deste apêndice apresenta trabalhos relacionados com cibersegurança e RDS. Também são apresentadas algumas semelhanças e diferenças entre esses trabalhos relacionados e a presente tese.

A.1 Cibersegurança e redes definidas por software

O conceito de RDS e o uso da tecnologia OpenFlow tem permitido novos tipos de aplicações para redes de computadores, inclusive em segurança (Braga *et al.*, 2010; Gutz *et al.*, 2012; Jafarian *et al.*, 2012; Liu *et al.*, 2011; Shin *et al.*, 2013b). Por exemplo, em Jesus *et al.* (2014) é apresentada uma análise do uso de RDS para manter-se a segurança de ambientes de nuvens.

Lara *et al.* (2013) verificaram que diferentes pesquisas têm explorado o controlador OpenFlow para propiciar um nível de inteligência aos *switches*, viabilizando novas formas de prover segurança. O OpenFlow possibilita a execução de tarefas administrativas ou de segurança sem a necessidade de acesso a toda topologia e tráfego de rede. Complementando suas percepções, o próprio OpenFlow provê informações e reações usadas em soluções clássicas de redes como o monitoramento e bloqueio de fluxos.

Nagahama *et al.* (2012) apresentaram a proposta do IPSFlow, uma aplicação em OpenFlow que viabiliza a captura seletiva e distribuída de tráfego para a análise em sistemas de detecção de intrusão. Segundo Nagahama *et al.* (2012), o IPSFlow permite o bloqueio de tráfego malicioso próximo à sua origem e incorpora características de detecção e reação a tráfegos maliciosos. A arquitetura proposta é semelhante a abordada neste trabalho quanto ao uso do OpenFlow e IPS, entretanto não foram realizados experimentos e validações e nem foi empregada CA. O trabalho apresenta uma abordagem restritiva, no caso o bloqueio de fluxo, o que pode comprometer os serviços de rede nos casos de falsos positivos.

Xing *et al.* (2013) apresentaram a proposta do SnortFlow, uma arquitetura para aprimorar os sistemas de prevenção de intrusão em ambientes de computação nas nuvens. Para cada máquina real há um IDS (Snort) para coletar informações de intrusão. Os alertas são enviados para um servidor responsável pela análise e geração das ações a serem aplicadas ao controlador. Os autores realizaram a validação de viabilidade do projeto e uma análise de desempenho em um cenário simplificado. Diferentemente, o presente trabalho não pretende analisar desempenho, mas verificar a efetividade e o impacto da combinação de IDS e OpenFlow em redes locais.

Shin *et al.* (2013a) desenvolveram um arcabouço para o desenvolvimento e execução de aplicações de segurança usando OpenFlow denominada de FRESCO. O arcabouço incorpora uma linguagem de *scripts* que possibilita a ligação de módulos por meio de compartilhamento de dados, eventos e a integração com ferramentas legadas de segurança, como por exemplo, IDS. Foram implementados dois casos de estudo: redirecionamento de varreduras maliciosas para um *honeypot* e a detecção de código malicioso usando aplicações de segurança legadas. Os resultados mostraram a viabilidade de composição de serviços e a simplicidade e tamanho reduzido dos *scripts* usando o arcabouço em relação a outras soluções. Os autores anunciaram que disponibilizarão o código

fonte futuramente, logo não é possível realizar comparações ou usar o arcabouço neste trabalho no momento.

Ballard *et al.* (2010) desenvolveram o OpenSAFE, uma solução para roteamento de tráfego por meio de um conjunto de abstrações, uma linguagem para especificar políticas e um controlador OpenFlow. A essência do OpenSAFE é viabilizar somente o processamento de fluxo de interesse, redirecionando ou copiando o fluxo para múltiplos filtros ou destinos, como por exemplo, IDS. Os autores alegam que em enlaces de alta velocidade é impossível o processamento ou cópia de todo o tráfego para ferramentas de monitoramento. Na solução apresenta nesta tese, a ideia é coletar o tráfego em posições estratégicas da rede por meio de múltiplos IDS. Além de monitorar, também são aplicadas ações preventivas baseadas no nível dos alertas produzidos pelos IDS.

Braga *et al.* (2010) apresentaram um mecanismo para monitoramento de ataques DDoS usando OpenFlow e Mapas Auto-Organizados. Suh *et al.* (2010) desenvolveram uma arquitetura que utiliza um agente no servidor para detectar DDoS por meio da análise de pacotes e que reage enviando uma notificação para um controlador OpenFlow, que por sua vez, alerta outros agentes para diminuir ou remover o tráfego de fluxos suspeitos para a máquina sob ataque. Comparado com esses trabalhos, adotamos uma abordagem na contagem de volumes combinada com alertas de IDS e não utilizamos agentes visando prover um ambiente que exige o mínimo de alteração para se utilizar a solução atômica. Apesar de usarmos um mecanismo simplificado de detecção, aplicamos medidas de controle de volume em tráfego suspeito antes de remover os fluxos, afetando assim, o volume de tráfego no início da suspeita de ataque.

Mehdi *et al.* (2011) comprovaram o papel importante de prover novas soluções de segurança a redes locais usando OpenFlow. Apresentaram a implementação de quatro algoritmos de detecção de anomalias em SOHO (*Small Office / Home Office*) usando OpenFlow. Comprovaram que a combinação de algoritmos com OpenFlow resulta em maior precisão na detecção de ameaças de segurança com relação a implementação em ISP (*Internet Service Provider*). Um resultado importante foi a constatação da maior precisão na detecção de ataques de varreduras de portas e DDoS, mesmo considerando diferentes taxas de fluxos de ataques injetados.

Em um contexto geral, o trabalho apresentado nesta tese se diferencia dos descritos anteriormente, por empregar CA, combinado com métodos de aprendizado de máquina não supervisionada, para então gerar regras de segurança autonômicas, fundamentadas no histórico de uso da rede e de alertas de segurança. O que permite que a rede se auto-proteja contra ciberameaças e também evita que a reação autonômica gere distúrbios em fluxos de rede não relacionados com ciberataques. Por fim, a tecnologia OpenFlow foi utilizada na arquitetura proposta por se mostrar uma ótima facilitadora na criação/manutenção de sensores e atuadores, o que é um ponto muito positivo e importante no desenvolvimento de sistemas autonômicos.

Referências Bibliográficas

Abie (2009) Habtamu Abie. Adaptive security and trust management for autonomic message-oriented middleware. Em *Mobile Adhoc and Sensor Systems, 2009. MASS '09. IEEE 6th International Conference on*, páginas 810–817. doi: 10.1109/MOBHOC.2009.5336915. Citado na pág. 48

Agrawal e Srikant (1994) Rakesh Agrawal e Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. Em *Proceedings of the 20th International Conference on Very Large Data Bases*, páginas 487–499, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. ISBN 1-55860-153-8. URL http://dl.acm.org/citation.cfm?id=645920.672836. Citado na pág. 69

Al-Nashif *et al.* (2008) Y. Al-Nashif, A.A. Kumar, S. Hariri, Guangzhi Qu, Yi Luo e F. Szidarovsky. Multi-level intrusion detection system (ml-ids). Em *Autonomic Computing, 2008. ICAC '08. International Conference on*, páginas 131–140. doi: 10.1109/ICAC.2008.25. Citado na pág. 54

Aljnidi e Leneutre (2007) Mohamad Aljnidi e Jean Leneutre. A security policy system for mobile autonomic networks. Em *Proceedings of the 1st international conference on Autonomic computing and communication systems*, Autonomics '07, páginas 23:1–23:9, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-09-7. URL http://dl.acm.org/citation.cfm?id=1365562.1365593. Citado na pág. 48

Alomari e Menasce (2012) F. Alomari e D. Menasce. An autonomic framework for integrating security and quality of service support in databases. Em *Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference on*, páginas 51–60. doi: 10.1109/SERE.2012.15. Citado na pág. 49

Alomari e Menascé (2013) Firas B. Alomari e Daniel A. Menascé. Self-protecting and self-optimizing database systems: implementation and experimental evaluation. Em *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, CAC '13, páginas 18:1–18:10, New York, NY, USA. ACM. ISBN 978-1-4503-2172-3. doi: 10.1145/2494621.2494631. URL http://doi.acm.org/10.1145/2494621.2494631. Citado na pág. 49

Ananthanarayanan *et al.* (2005) R. Ananthanarayanan, M. Mohania e A. Gupta. Management of conflicting obligations in self-protecting policy-based systems. Em *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, páginas 274–285. doi: 10.1109/ICAC.2005.38. Citado na pág. 48

Anderson (2012) Devin Anderson. Three steps to ensuring byod doesn't lead to byot (bring your own threat), 2012. URL http://www.scmagazineuk.com/three-steps-to-ensuring-byod-doesnt-lead-to-byot-bring-your-own-threat/article/222272/. Acessado em 08/07/2014. Disponível em http://www.scmagazineuk.com/. Citado na pág. 2, 20

Apache (2016a) Apache. The apache http server project, 2016a. URL https://httpd.apache.org/. Acessado em 24/04/2016. Disponível em https://httpd.apache.org/. Citado na pág. 104

Apache (2016b) Apache. Apache lucene, 2016b. URL https://lucene.apache.org/. Acessado em 04/02/2016. Disponível em https://lucene.apache.org/. Citado na pág. 92

Apache (2016c) Apache. Apache license 2.0, 2016c. URL http://www.apache.org/licenses/LICENSE-2.0. Acessado em 30/01/2016. Disponível em http://www.apache.org/licenses/LICENSE-2.0. Citado na pág. 89

Armstrong *et al.* (2003) D. Armstrong, G. Frazier, S. Carter e T. Frazier. A controller-based autonomic defense system. Em *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, volume 2, páginas 21–23 vol.2. doi: 10.1109/DISCEX.2003.1194902. Citado na pág. 52

Arora *et al.* (2006) H. Arora, B.K. Mishra e T. S. Raghu. Autonomic-computing approach to secure knowledge management: a game-theoretic analysis. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 36(3):487–497. ISSN 1083-4427. doi: 10.1109/TSMCA.2006.871724. Citado na pág. 53

Ashby (1960) William Ross. Ashby. *Design for a brain : the origin of adaptive behaviour / [by] W. Ross Ashby.* Chapman and Hall London, 2nd ed., rev. ed. Citado na pág. xi, 23, 24

Baddeley (1997) A.D. Baddeley. *Human Memory: Theory and Practice*. Psychology Press. ISBN 9780863774317. URL https://books.google.com.br/books?id=fMgm-2NXAXYC. Citado na pág. 73

Bailey (2012) Christopher Bailey. Application of self-adaptive techniques to federated authorization models. Em *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, páginas 1495–1498, Piscataway, NJ, USA. IEEE Press. ISBN 978-1-4673-1067-3. URL http://dl.acm.org/citation.cfm?id=2337223.2337465. Citado na pág. 48

Ballard *et al.* (2010) Jeffrey R. Ballard, Ian Rae e Aditya Akella. Extensible and scalable network monitoring using opensafe. Em *Proceedings of INM/WREN*. Citado na pág. 151

Barayuga e Yu (2015) V. J. D. Barayuga e W. E. S. Yu. Packet level tcp performance of nat44, nat64 and ipv6 using iperf in the context of ipv6 migration. Em *IT Convergence and Security (ICITCS), 2015 5th International Conference on*, páginas 1–3. doi: 10.1109/ICITCS.2015.7293006. Citado na pág. 98

Barnyard2 (2015) Barnyard2. Barnyard 2, 2015. Acessado em 12/08/2015. Disponível em https://github.com/firnsy/barnyard2. Citado na pág. 66

Barrère *et al.* (2011) Martín Barrère, Rémi Badonnel e Olivier Festor. Supporting vulnerability awareness in autonomic networks and systems with oval. Em *Proceedings of the 7th International Conference on Network and Services Management*, CNSM '11, páginas 37–45, Laxenburg, Austria, Austria. International Federation for Information Processing. ISBN 978-3-901882-44-9. URL http://dl.acm.org/citation.cfm?id=2147671.2147678. Citado na pág. 49, 50

Barrère *et al.* (2012) M. Barrère, R. Badonnel e O. Festor. Collaborative remediation of configuration vulnerabilities in autonomic networks and systems. Em *Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm)*, páginas 357–363. Citado na pág. 50

Barrère *et al.* (2014) Martin Barrère, Remi Badonnel e Olivier Festor. Vulnerability assessment in autonomic networks and services: A survey. *Communications Surveys Tutorials, IEEE*, 16(2):988–1004. ISSN 1553-877X. doi: 10.1109/SURV.2013.082713.00154. Citado na pág. 50, 55

Bashir e Chachoo (2014) Uzair Bashir e Manzoor Chachoo. Intrusion detection and prevention system: Challenges amp; opportunities. Em *Computing for Sustainable Global Development (INDIACom), 2014 International Conference on*, páginas 806–809. doi: 10.1109/IndiaCom.2014.6828073. Citado na pág. 15, 16

Batista *et al.* **(2016)** Daniel M. Batista, Luiz A. F. Santos, Rodrigo Campiolo, Wagner A. Monteverde, Marlon F. Antonio, Thiago L. Vieira, Eder Ferreira, Rafael Silvério e Fausto Vetter. Gt-ews: Building a cybersecurity ews based on social networks. *TNC - Networking Conference*. URL https://tnc16.geant.org/core/presentation/707. Citado na pág. 139

Berral *et al.* **(2008)** Josep L. Berral, Nicolas Poggi, Javier Alonso, Ricard Gavaldà, Jordi Torres e Manish Parashar. Adaptive distributed mechanism against flooding network attacks based on machine learning. Em *Proceedings of the 1st ACM workshop on Workshop on AISec*, AISec '08, páginas 43–50, New York, NY, USA. ACM. ISBN 978-1-60558-291-7. doi: 10.1145/1456377.1456389. URL http://doi.acm.org/10.1145/1456377.1456389. Citado na pág. 51

Bishop (2004) Matt Bishop. *Introduction to Computer Security*. Addison-Wesley Professional. ISBN 0321247442. Citado na pág. 5, 6, 7, 8, 11, 12, 13, 14, 15, 16

Braga *et al.* **(2010)** Rodrigo Braga, Edjard Mota e Alexandre Passito. Lightweight ddos flooding attack detection using nox/openflow. Em *Proceedings of the 2010 IEEE 35th Conference on Local Computer Networks*, LCN '10, páginas 408–415, Washington, DC, USA. ISBN 978-1-4244-8387-7. doi: 10.1109/LCN.2010.5735752. URL http://dx.doi.org/10.1109/LCN.2010.5735752. Citado na pág. 150, 151

Campiolo (2016) Rodrigo Campiolo. *Evaluating the utilization of Twitter messages as a source of security alerts*. Tese de Doutorado, USP. Citado na pág. 136, 137, 139

Campiolo *et al.* **(2013)** Rodrigo Campiolo, Luiz A. F. Santos, Daniel M. Batista e Marco A. Gerosa. Uma abordagem colaborativa e distribuída para a detecção antecipada de incidentes de segurança usando fontes de dados heterogêneas e aberta. Em *28th ACM SAC*. Citado na pág. 59, 87, 89, 90, 93, 135, 136

Cha *et al.* **(2012)** M. Cha, F. Benevenuto, H. Haddadi e K. Gummadi. The world of connections and information flow in twitter. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, PP(99):1 –8. ISSN 1083-4427. doi: 10.1109/TSMCA.2012.2183359. Citado na pág. 87

Chang *et al.* **(2013)** Jian Chang, Krishna K. Venkatasubramanian, Andrew G. West e Insup Lee. Analyzing and defending against web-based malware. *ACM Comput. Surv.*, 45(4):49:1–49:35. ISSN 0360-0300. doi: 10.1145/2501654.2501663. URL http://doi.acm.org/10.1145/2501654.2501663. Citado na pág. 5, 7, 12, 13, 16

Chen *et al.* **(2013)** Qian Chen, Sherif Abdelwahed e Abdelkarim Erradi. A model-based approach to self-protection in computing system. Em *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, CAC '13, páginas 16:1–16:10, New York, NY, USA. ACM. ISBN 978-1-4503-2172-3. doi: 10.1145/2494621.2494639. URL http://doi.acm.org/10.1145/2494621.2494639. Citado na pág. 54, 55

Chen *et al.* **(2014)** Qian Chen, S. Abdelwahed e A. Erradi. A model-based validated autonomic approach to self-protect computing systems. *IEEE Internet of Things Journal*. ISSN 2327-4662. doi: 10.1109/JIOT.2014.2349899. Citado na pág. 55, 145

Chess (2005) David M. Chess. Security in autonomic computing. *SIGARCH Comput. Archit. News*, 33(1):2–5. ISSN 0163-5964. doi: 10.1145/1055626.1055628. URL http://doi.acm.org/10.1145/1055626.1055628. Citado na pág. 48

Chess *et al.* **(2003)** D.M. Chess, C.C. Palmer e S.R. White. Security in an autonomic computing environment. *IBM Systems Journal*, 42(1):107–118. ISSN 0018-8670. doi: 10.1147/sj.421.0107. Citado na pág. 48

Chiang *et al.* **(2007)** F. Chiang, J. Agbinya e R. Braun. Risk and vulnerability assessment of secure autonomic communication networks. Em *Wireless Broadband and Ultra Wideband Communications, 2007. AusWireless 2007. The 2nd International Conference on*, páginas 40–40. doi: 10.1109/AUSWIRELESS.2007.67. Citado na pág. 53, 54

Chu *et al.* **(2010)** YuHunag Chu, MinChi Tseng, YaoTing Chen, YuChieh Chou e YanRen Chen. A novel design for future on-demand service and security. Em *Communication Technology (ICCT), 2010 12th IEEE International Conference on*, páginas 385–388. doi: 10.1109/ICCT.2010.5689156. Citado na pág. 55, 145

Cisco (2014) Cisco. Cisco 2014 anual security report, 2014. URL https://www.cisco.com/web/offer/gist_ty2_asset/Cisco_2014_ASR.pdf. Acessado em 30/06/2014. Disponível em https://www.cisco.com/. Citado na pág. 5, 12, 13, 18, 20

Cisco (2016) Cisco. Cisco 2016 anual security report, 2016. Acessado em 14/07/2016. Disponível em https://www.cisco.com/. Citado na pág. 1, 18

Costa *et al.* **(2016)** Leonardo C. Costa, Alex B. Vieira, Erik B Silva, Daniel F. Macedo, Geraldo Gomes, Luiz H. A. Correia e Luiz F. M. Vieira. Avaliação de desempenho de planos de dados openflow. Em *SBRC 2016*, Salvador, Bahia. URL http://www.sbrc2016.ufba.br/downloads/anais/proceedingsSBRC2016.pdf. Citado na pág. 145

Dai *et al.* **(2006)** Yuan-Shun Dai, M. Hinchey, Mingrui Qi e Xukai Zou. Autonomic security and self-protection based on feature-recognition with virtual neurons. Em *Dependable, Autonomic and Secure Computing, 2nd IEEE International Symposium on*, páginas 227–234. doi: 10.1109/DASC.2006.24. Citado na pág. 53

de Azevedo *et al.* **(2009)** R.R. de Azevedo, E.R.G. Dantas, R.C. dos Santos, C. Rodrigues, F. Freitas e M.J. Siqueira. An autonomic multiagent system ontology-based for management of security of information. Em *Internet Technology and Secured Transactions, 2009. ICITST 2009. International Conference for*, páginas 1–9. Citado na pág. 52

Disterer e Kleiner (2013) Georg Disterer e Carsten Kleiner. Using mobile devices with byod. *Int. J. Web Portals*, 5(4):33–45. ISSN 1938-0194. doi: 10.4018/ijwp.2013100103. URL http://dx.doi.org/10.4018/ijwp.2013100103. Citado na pág. 2, 19

Doria *et al.* **(2010)** A. Doria, J. Hadi Salim, R. Haas, W. Wang, L. Dong e R. Gopal. Forwarding and control element separation (forces) protocol specification, 2010. URL http://tools.ietf.org/html/rfc5810. Acessado em 20/05/2014. Disponível em http://tools.ietf.org/html/rfc5810. Citado na pág. 31

Egele *et al.* **(2008)** Manuel Egele, Theodoor Scholte, Engin Kirda e Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput. Surv.*, 44 (2):6:1–6:42. ISSN 0360-0300. doi: 10.1145/2089125.2089126. URL http://doi.acm.org/10.1145/2089125.2089126. Citado na pág. 16

Erickson (2013) David Erickson. The beacon openflow controller. Em *Proceedings of 2th ACM SIGCOMM*, HotSDN '13, páginas 13–18, NY, USA. ACM. ISBN 978-1-4503-2178-5. doi: 10.1145/2491185.2491189. URL http://doi.acm.org/10.1145/2491185.2491189. Citado na pág. xi, 45, 46, 98

Erickson (2014) David Erickson. Beacon - confluence, 2014. URL https://openflow.stanford.edu/display/Beacon/Home. Acessado em 15/06/2014. Disponível em https://openflow.stanford.edu/display/Beacon/Home. Citado na pág. xi, 45, 47, 66

Esposte *et al.* **(2016)** Arthur Esposte, Rodrigo Campiolo, Fabio Kon e Daniel Batista. A collaboration model to recommend network security alerts based on the mixed hybrid approach. Em *SBRC*, Salvador, Bahia. URL http://XXXXX/152330.pdf. Citado na pág. 139

Feily *et al.* (2009) M. Feily, A. Shahrestani e S. Ramadass. A survey of botnet and botnet detection. Em *Emerging Security Information, Systems and Technologies, 2009. SECURWARE '09. Third International Conference on*, páginas 268–273. doi: 10.1109/SECURWARE.2009.48. Citado na pág. 1, 8, 9, 11, 13

Floodlight (2014) Floodlight. Floodlight openflow controller, 2014. URL http://www.projectfloodlight.org/floodlight/. Acessado em 15/06/2014. Disponível em http://www.projectfloodlight.org/floodlight/. Citado na pág. 45

Flowgrammable (2014) Flowgrammable. Sdn / openflow, 2014. URL http://flowgrammable.org/. Acessado em 27/05/2014. Disponível em http://flowgrammable.org/sdn/openflow/. Citado na pág. xi, xiii, 34, 36, 37, 38, 39, 40, 42, 44

Flowvisor (2014) Flowvisor. Flowvisor, 2014. URL https://openflow.stanford.edu/display/DOCS/Flowvisor. Acessado em 15/06/2014. Disponível em https://openflow.stanford.edu/display/DOCS/Flowvisor. Citado na pág. 45

Fournier-Viger (2014) Philippe Fournier-Viger. Spmf - an open-source data mining library, 2014. URL http://www.philippe-fournier-viger.com/spmf/. Acessado em 09/12/2014. Disponível em http://www.philippe-fournier-viger.com/spmf/index.php?link=performance.php. Citado na pág. 70

Gnip (2016) Gnip. Gnip twitter, 2016. Acessado em 28/01/2016. Disponível em https://gnip.com/sources/twitter/. Citado na pág. 89

Guan e Fu (2010) Qiang Guan e Song Fu. auto-aid: A data mining framework for autonomic anomaly identification in networked computer systems. Em *Performance Computing and Communications Conference (IPCCC), 2010 IEEE 29th International*, páginas 73–80. doi: 10.1109/PCCC.2010.5682334. Citado na pág. 51

Gutz *et al.* (2012) Stephen Gutz, Alec Story, Cole Schlesinger e Nate Foster. Splendid isolation: A slice abstraction for software-defined networks. Em *Proceedings of the HotSDN'12*, páginas 79–84, New York, NY, USA. ACM. ISBN 978-1-4503-1477-0. doi: 10.1145/2342441.2342458. URL http://doi.acm.org/10.1145/2342441.2342458. Citado na pág. 150

Han *et al.* (2004) Jiawei Han, Jian Pei, Yiwen Yin e Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8 (1):53–87. ISSN 1384-5810. doi: 10.1023/B:DAMI.0000005258.31418.83. URL http://dx.doi.org/10.1023/B:DAMI.0000005258.31418.83. Citado na pág. 68, 69, 70

Hariri *et al.* (2006) Salim Hariri, Bithika Khargharia, Houping Chen, Jingmei Yang, Yeliang Zhang, Manish Parashar e Hua Liu. The autonomic computing paradigm. *Cluster Computing*, 9(1):5–17. ISSN 1386-7857. doi: 10.1007/s10586-006-4893-0. URL http://dx.doi.org/10.1007/s10586-006-4893-0. Citado na pág. xi, 2, 21, 23, 24, 25

Harrington (2012) Peter Harrington. *Machine Learning in Action*. Manning Publications Co., Greenwich, CT, USA. ISBN 1617290181, 9781617290183. Citado na pág. xi, 69, 70, 72, 104

He *et al.* (2010a) Ruan He, M. Lacoste e J. Leneutre. Virtual security kernel: A component-based os architecture for self-protection. Em *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, páginas 851–858. doi: 10.1109/CIT.2010.160. Citado na pág. 48

He *et al.* (2010b) Ruan He, M. Lacoste e J. Leneutre. A policy management framework for self-protection of pervasive systems. Em *Autonomic and Autonomous Systems (ICAS), 2010 Sixth International Conference on*, páginas 104–109. doi: 10.1109/ICAS.2010.22. Citado na pág. 48

Hping3 (2016) Hping3. Hping security tool, 2016. URL http://www.hping.org/hping3.html. Acessado em 25/04/2016. Disponível em http://www.hping.org/hping3.html. Citado na pág. 104

Huebscher e McCann (2008) Markus C. Huebscher e Julie A. McCann. A survey of autonomic computing - degrees, models, and applications. *ACM Comput. Surv.*, 40(3):7:1–7:28. ISSN 0360-0300. doi: 10.1145/1380584.1380585. URL http://doi.acm.org/10.1145/1380584.1380585. Citado na pág. 2, 21, 22, 25, 27

Hunyadi (2011) Daniel Hunyadi. Performance comparison of apriori and fp-growth algorithms in generating association rules. Em *Proceedings of the 5th European Conference on European Computing Conference*, ECC'11, páginas 376–381, Stevens Point, Wisconsin, USA. World Scientific and Engineering Academy and Society (WSEAS). ISBN 978-960-474-297-4. URL http://dl.acm.org/citation.cfm?id=1991016.1991084. Citado na pág. 70

HYENAE (2016) HYENAE. Hyenae, 2016. URL https://sourceforge.net/projects/hyenae/. Acessado em 25/02/2016. Disponível em https://sourceforge.net/projects/hyenae/. Citado na pág. 101

IBM (2003) IBM. An architectural blueprint for autonomic computing. Relatório Técnico April, IBM. Citado na pág. 28, 48

IBM (2006) IBM. An architectural blueprint for autonomic computing . *Quality*, 36(June): 34. URL http://users.encs.concordia.ca/~ac/ac-resources/AC_Blueprint_White_Paper_4th.pdf. Citado na pág. xi, 23, 25, 26, 27, 28, 29

IDSwakeup (2016) IDSwakeup. Hsc - tools - idswakeup, 2016. URL http://www.hsc.fr/ressources/outils/idswakeup/index.html.en. Acessado em 15/05/2016. Disponível em http://www.hsc.fr/ressources/outils/idswakeup/index.html.en. Citado na pág. 124, 125

IEEE (2008) IEEE. 802.1ah - provider backbone bridges, 2008. URL http://www.ieee802.org/1/pages/802.1ah.html. Acessado em 31/05/2014. Disponível em http://www.ieee802.org/1/pages/802.1ah.html. Citado na pág. 38

IETF (2002) IETF. An introduction to the stream control transmission protocol (sctp), 2002. URL http://tools.ietf.org/html/rfc3286. Acessado em 29/05/2014. Disponível em http://tools.ietf.org/html/rfc3286. Citado na pág. 34

Iperf (2016) Iperf. Iperf - the tcp, udp and sctp network bandwidth measurement tool, 2016. URL https://iperf.fr/. Acessado em 25/06/2016. Disponível em https://iperf.fr/. Citado na pág. 98

Ippoliti e Zhou (2012) Dennis Ippoliti e Xiaobo Zhou. A self-tuning self-optimizing approach for automated network anomaly detection systems. Em *Proceedings of the 9th international conference on Autonomic computing*, ICAC '12, páginas 85–90, New York, NY, USA. ACM. ISBN 978-1-4503-1520-3. doi: 10.1145/2371536.2371551. URL http://doi.acm.org/10.1145/2371536.2371551. Citado na pág. 51

ISO27001 (2013) ISO27001. Iso/iec 27001:2013 - information technology - security techniques - information security management systems - requirements, 2013. URL https://www.iso.org/obp/ui/#iso:std:iso-iec:27001:ed-2:v1:en. Acessado em 20/06/2014. Disponível em https://www.iso.org/obp/ui/#iso:std:iso-iec:27001:ed-2:v1:en. Citado na pág. 5

Jafarian *et al.* (2012) Jafar Haadi Jafarian, Ehab Al-Shaer e Qi Duan. Openflow random host mutation: Transparent moving target defense using software defined networking. Em *Proceedings of HotSDN '12*, páginas 127–132, New York, NY, USA. ACM. ISBN 978-1-4503-1477-0. doi: 10.1145/2342441.2342467. URL http://doi.acm.org/10.1145/2342441.2342467. Citado na pág. 150

Jesus *et al.* (2014) Wanderson Paim de Jesus, Daniel Alves da Silva, Rafael T. de Sousa Júnior e Francisco Vitor Lopes da Frota. Analysis of sdn contributions for cloud computing security. Em *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, UCC '14, páginas 922–927, Washington, DC, USA. IEEE Computer Society. ISBN

978-1-4799-7881-6. doi: 10.1109/UCC.2014.150. URL http://dx.doi.org/10.1109/UCC.2014.150. Citado na pág. 144, 150

Karresand (2003) M. Karresand. Separating trojan horses, viruses, and worms - a proposed taxonomy of software weapons. Em *Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society*, páginas 127–134. doi: 10.1109/SMCSIA.2003.1232411. Citado na pág. 7, 8

Kaspersky (2012) Kaspersky. Blackhole: a mais noba arma dos cibercriminosos brasileiros, 2012. URL http://brazil.kaspersky.com/sobre-a-kaspersky/centro-de-imprensa/blog-da-kaspersky/exploit-kits-br. Acessado em 30/06/2014. Disponível em http://brazil.kaspersky.com/. Citado na pág. 19

Kaspersky (2013) Kaspersky. Kaspersky security bulletin 2013, 2013. URL http://media.kaspersky.com/pdf/KSB_2013_EN.pdf. Acessado em 30/06/2014. Disponível em http://media.kaspersky.com/. Citado na pág. 12, 18, 20

Kaspersky (2015) Kaspersky. Kaspersky security bulletin 2015, 2015. Acessado em 13/07/2016. Disponível em http://media.kaspersky.com/. Citado na pág. 18

Kayem *et al.* (2008) A.V.D.M. Kayem, P. Martin, S.G. Akl e W. Powley. A framework for self-protecting cryptographic key management. Em *Self-Adaptive and Self-Organizing Systems, 2008. SASO '08. Second IEEE International Conference on*, páginas 191–200. doi: 10.1109/SASO.2008.57. Citado na pág. 48

Kephart e Chess (2003) J.O. Kephart e D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50. ISSN 0018-9162. doi: 10.1109/MC.2003.1160055. Citado na pág. 2, 21, 48

Kosters *et al.* (2003) Walter A. Kosters, Wim Pijls e Viara Popova. Complexity analysis of depth first and fp-growth implementations of apriori. Em *Proceedings of the 3rd International Conference on Machine Learning and Data Mining in Pattern Recognition*, MLDM'03, páginas 284–292, Berlin, Heidelberg. Springer-Verlag. ISBN 3-540-40504-6. URL http://dl.acm.org/citation.cfm?id=1759548.1759580. Citado na pág. 70

Kumar e Selvakumar (2009) P.A.R. Kumar e S. Selvakumar. Distributed denial-of-service (ddos) threat in collaborative environment - a survey on ddos attack tools and traceback mechanisms. Em *Advance Computing Conference, 2009. IACC 2009. IEEE International*, páginas 1275–1280. doi: 10.1109/IADCC.2009.4809199. Citado na pág. 9, 10

Lara *et al.* (2013) A. Lara, A. Kolasani e B. Ramamurthy. Network innovation using openflow: A survey. *Communications Surveys Tutorials, IEEE*, PP(99):1–20. ISSN 1553-877X. doi: 10.1109/SURV.2013.081313.00105. Citado na pág. 40, 43, 150

Lara *et al.* (2014) A. Lara, A. Kolasani e B. Ramamurthy. Network innovation using openflow: A survey. *Communications Surveys Tutorials, IEEE*, 16(1):493–512. ISSN 1553-877X. doi: 10.1109/SURV.2013.081313.00105. Citado na pág. 29, 30, 31

Lee *et al.* (2011) Jae Woo Lee, R. Francescangeli, J. Janak, S. Srinivasan, S.A. Baset, Henning Schulzrinne, Z. Despotovic e W. Kellerer. Netserv: Active networking 2.0. Em *Communications Workshops (ICC), 2011 IEEE International Conference on*, páginas 1–6. doi: 10.1109/iccw.2011.5963554. Citado na pág. 31

Lerman e Ghosh (2010) Kristina Lerman e Rumi Ghosh. Information contagion: an empirical study of the spread of news on digg and twitter social networks. *CoRR*, abs/1003.2664. Citado na pág. 87

Li *et al.* **(2008)** Bingyang Li, Huiqiang Wang e Guangsheng Feng. Adaptive hierarchical intrusion tolerant model based on autonomic computing. Em *Security Technology, 2008. SECTECH '08. International Conference on*, páginas 137–141. doi: 10.1109/SecTech.2008.24. Citado na pág. 54

Li *et al.* **(2010)** Bingyang Li, Xingyuan Zhang e Fei Xie. Modeling and analyzing of autonomic intrusion tolerant based on pepa. Em *Computer and Information Application (ICCIA), 2010 International Conference on*, páginas 244–247. doi: 10.1109/ICCIA.2010.6141582. Citado na pág. 54

Liu *et al.* **(2011)** Xiong Liu, Haiwei Xue, Xiaoping Feng e Yiqi Dai. Design of the multi-level security network switch system which restricts covert channel. Em *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*, páginas 233–237. doi: 10.1109/ICCSN.2011.6013582. Citado na pág. 150

Loukas e Oke (2007) G. Loukas e G. Oke. A biologically inspired pired denial of service detector using the random neural network. Em *Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE Internatonal Conference on*, páginas 1–6. doi: 10.1109/MOBHOC.2007.4428683. Citado na pág. 51

Macías-Escrivá *et al.* **(2013)** Frank D. Macías-Escrivá, Rodolfo Haber, Raul del Toro e Vicente Hernandez. Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2013.07.033. URL http://www.sciencedirect.com/science/article/pii/S0957417413005125. Citado na pág. 22

Maestro (2014) Maestro. maestro-platform: A scalable control platform writen in java which supports openflow, 2014. URL https://code.google.com/p/maestro-platform/. Acessado em 15/06/2014. Disponível em https://code.google.com/p/maestro-platform/. Citado na pág. 45

Mairh *et al.* **(2011)** Abhishek Mairh, Debabrat Barik, Kanchan Verma e Debasish Jena. Honeypot in network security: A survey. Em *Proceedings of the 2011 International Conference on Communication, Computing & Security*, ICCCS '11, páginas 600–605, New York, NY, USA. ACM. ISBN 978-1-4503-0464-1. doi: 10.1145/1947940.1948065. URL http://doi.acm.org/10.1145/1947940.1948065. Citado na pág. 16

Maliki e Seigneur (2010) T.E. Maliki e J. M Seigneur. A security adaptation reference monitor (sarm) for highly dynamic wireless environments. Em *Emerging Security Information Systems and Technologies (SECURWARE), 2010 Fourth International Conference on*, páginas 63–68. doi: 10.1109/SECURWARE.2010.18. Citado na pág. 49

Manning *et al.* **(2008)** Christopher D. Manning, Prabhakar Raghavan e Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK. ISBN 978-0-521-86571-5. Citado na pág. 87, 92

Marnerides *et al.* **(2008)** Angelos Marnerides, Dimitrios P. Pezaros e David Hutchison. Detection and mitigation of abnormal traffic behaviour in autonomic networked environments. Em *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, páginas 51:1–51:2, New York, NY, USA. ACM. ISBN 978-1-60558-210-8. doi: 10.1145/1544012.1544063. URL http://doi.acm.org/10.1145/1544012.1544063. Citado na pág. 50

Marpaung *et al.* **(2012)** J.A.P. Marpaung, M. Sain e Hoon-Jae Lee. Survey on malware evasion techniques: State of the art and challenges. Em *Advanced Communication Technology (ICACT), 2012 14th International Conference on*, páginas 744–749. Citado na pág. 7, 13

McKeown *et al.* **(2008)** Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker e Jonathan Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74. ISSN 0146-4833. doi:

10.1145/1355734.1355746. URL http://doi.acm.org/10.1145/1355734.1355746. Citado na pág. xi, 2, 30, 31, 32, 43

Mehdi *et al.* **(2011)** Syed Akbar Mehdi, Junaid Khalid e Syed Ali Khayam. Revisiting traffic anomaly detection using software defined networking. Em *Proceedings of RAID'11*, páginas 161–180. ISBN 978-3-642-23643-3. doi: 10.1007/978-3-642-23644-0_9. Citado na pág. 151

Meng *et al.* **(2015)** Guozhu Meng, Yang Liu, Jie Zhang, Alexander Pokluda e Raouf Boutaba. Collaborative security: A survey and taxonomy. *ACM Comput. Surv.*, 48(1):1:1–1:42. ISSN 0360-0300. doi: 10.1145/2785733. URL http://doi.acm.org/10.1145/2785733. Citado na pág. 1

Merideth e Narasimhan (2005) Michael G. Merideth e Priya Narasimhan. Retrofitting networked applications to add autonomic reconfiguration. *SIGSOFT Softw. Eng. Notes*, 30(4):1–7. ISSN 0163-5948. doi: 10.1145/1082983.1083073. URL http://doi.acm.org/10.1145/1082983.1083073. Citado na pág. 53

Miller *et al.* **(2012)** Keith W. Miller, Jeffrey Voas e George F. Hurlburt. Byod: Security and privacy considerations. *IT Professional*, 14(5):53–55. ISSN 1520-9202. doi: 10.1109/MITP.2012.93. URL http://dx.doi.org/10.1109/MITP.2012.93. Citado na pág. 19, 20

Morris *et al.* **(2012)** Meredith Ringel Morris, Scott Counts, Asta Roseway, Aaron Hoff e Julia Schwarz. Tweeting is believing?: understanding microblog credibility perceptions. Em *Proceedings of the ACM CSCW*, páginas 441–450. ISBN 978-1-4503-1086-4. doi: 10.1145/2145204.2145274. Citado na pág. 94

Nagahama *et al.* **(2012)** Fábio Yu Nagahama, Fernando Farias, Elisângela Aguiar, Luciano Gaspary, Lisandro Granville, Eduardo Cerqueira e Antônio Abelém. Ipsflow – uma proposta de sistema de prevenção de intrusão baseado no framework openflow. Em *Anais do XII SBSeg*, páginas 324–330. Citado na pág. 3, 150

Nalavade e Meshram (2010) K. C. Nalavade e B. B. Meshram. Intrusion prevention systems: Data mining approach. Em *Proceedings of the International Conference and Workshop on Emerging Trends in Technology*, ICWET '10, páginas 211–214, New York, NY, USA. ACM. ISBN 978-1-60558-812-4. doi: 10.1145/1741906.1741952. URL http://doi.acm.org/10.1145/1741906.1741952. Citado na pág. 15

Nami e Bertels (2007) M.R. Nami e K. Bertels. A survey of autonomic computing systems. Em *Autonomic and Autonomous Systems, 2007. ICAS07. Third International Conference on*, páginas 26–26. doi: 10.1109/CONIELECOMP.2007.48. Citado na pág. xi, 23, 27, 28

NOX (2014) NOX. About nox, 2014. URL http://www.noxrepo.org/nox/about-nox/. Acessado em 15/06/2014. Disponível em http://www.noxrepo.org/nox/about-nox/. Citado na pág. 45

Nunes *et al.* **(2014)** B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka e T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks, 2014. ISSN 1553-877X. Citado na pág. xi, 29, 30, 31, 33, 40, 43, 44

OISFoundation (2013) OISFoundation. Suricata - open source ids / ips / nsm engine, 2013. URL http://suricata-ids.org. Acessado em 17/11/2013. Disponível em http://suricata-ids.org. Citado na pág. 62

Olvera-Irigoyen *et al.* **(2011)** O. Olvera-Irigoyen, A. Kortebi, L. Toutain e D. Ros. Available bandwidth probing in hybrid home networks. Em *Local Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on*, páginas 1–7. doi: 10.1109/LANMAN.2011.6076944. Citado na pág. 98

ONF (2011) ONF. Openflow 1.2, 2011. URL https://www.opennetworking.org/. Acessado em 02/06/2014. Disponível em https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf. Citado na pág. 33

ONF (2012) ONF. Open networking foundation: Openflow switch specification - version 1.3.0, 2012. URL https://www.opennetworking.org/. Acessado em 02/06/2014. Disponível em https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf. Citado na pág. xi, xiii, 33, 34, 36, 37, 40, 41

ONF (2013) ONF. Open networking foundation: Openflow switch specification - version 1.4.0, 2013. URL https://www.opennetworking.org/. Acessado em 02/06/2014. Disponível em https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf. Citado na pág. xi, 33, 35, 37, 40, 44

ONF (2014) ONF. Open networking foundation, 2014. URL https://www.opennetworking.org/. Acessado em 17/15/2014. Disponível em https://www.opennetworking.org/. Citado na pág. 31

OpenFlow (2011) OpenFlow. Openflow switch specification - version 1.1.0 implemented, 2011. URL http://archive.openflow.org/documents/. Acessado em 26/05/2014. Disponível em http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf. Citado na pág. xi, xiii, 32, 33, 35, 36, 38, 39

Paradis (2014) Thomas Paradis. Software-defined networking. Dissertação de Mestrado, KTH, School of Information and Communication Technology (ICT). Citado na pág. 29, 30

Patel *et al.* (2012) Ahmed Patel, Mona Taghavi, Kaveh Bakhtiyari e Joaquim Celestino Júnior. Taxonomy and proposed architecture of intrusion detection and prevention systems for cloud computing. Em *Proceedings of the 4th International Conference on Cyberspace Safety and Security*, CSS'12, páginas 441–458, Berlin, Heidelberg. Springer-Verlag. ISBN 978-3-642-35361-1. doi: 10.1007/978-3-642-35362-8_33. URL http://dx.doi.org/10.1007/978-3-642-35362-8_33. Citado na pág. 15

Paxson (2013) V. et al. Paxson. The bro network security monitor, 2013. URL http://www.bro.org. Acessado em 17/11/2013. Disponível em http://www.bro.org. Citado na pág. 62

Peng *et al.* (2007) Tao Peng, Christopher Leckie e Kotagiri Ramamohanarao. Survey of network-based defense mechanisms countering the dos and ddos problems. *ACM Comput. Surv.*, 39 (1). ISSN 0360-0300. doi: 10.1145/1216370.1216373. URL http://doi.acm.org/10.1145/1216370.1216373. Citado na pág. 9, 10, 11

Percoco (2013) Nicholas J. Percoco. 2013 global security report. Relatório técnico, Trustwave Holdings Inc. Citado na pág. 1

Phuvipadawat e Murata (2010) S. Phuvipadawat e T. Murata. Breaking news detection and tracking in twitter. Em *Proc. of the IEEE/WIC/ACM WI-IAT*, volume 3, páginas 120 –123. doi: 10.1109/WI-IAT.2010.205. Citado na pág. 95

PostGreSQL (2015) PostGreSQL. Postgresql - the world's most advanced open source database, 2015. URL http://www.postgresql.org/. Acessado em 12/08/2015. Disponível em http://www.postgresql.org/. Citado na pág. 66

POX (2014) POX. About pox, 2014. URL http://www.noxrepo.org/pox/about-pox/. Acessado em 15/06/2014. Disponível em http://www.noxrepo.org/pox/about-pox/. Citado na pág. 45

Qu *et al.* (2004) Guangzhi Qu, S. Hariri, S. Jangiti, J. Rudraraju, Seungchan Oh, S. Fayssal, Guangsen Zhang e M. Parashar. Online monitoring and analysis for self-protection against network attacks. Em *Autonomic Computing, 2004. Proceedings. International Conference on*, páginas 324–325. doi: 10.1109/ICAC.2004.1301398. Citado na pág. 52

Qu *et al.* (2011) Yan Qu, Chen Huang, Pengyi Zhang e Jun Zhang. Microblogging after a major disaster in China: a case study of the 2010 yushu earthquake. Em *Proceedings of the ACM CSCW*, páginas 25–34. ISBN 978-1-4503-0556-3. doi: 10.1145/1958824.1958830. Citado na pág. 95

Rao e Pati (2012) U.H. Rao e B.P. Pati. Study of internet security threats among home users. Em *Computational Aspects of Social Networks (CASoN), 2012 Fourth International Conference on*, páginas 217–221. doi: 10.1109/CASoN.2012.6412405. Citado na pág. 5

Rawat e Saxena (2009) Sanjay Rawat e Ashutosh Saxena. Danger theory based syn flood attack detection in autonomic network. Em *Proceedings of the 2nd international conference on Security of information and networks*, SIN '09, páginas 213–218, New York, NY, USA. ACM. ISBN 978-1-60558-412-6. doi: 10.1145/1626195.1626248. URL http://doi.acm.org/10.1145/1626195.1626248. Citado na pág. 52

Rodríguez-Gómez *et al.* (2013) Rafael A. Rodríguez-Gómez, Gabriel Maciá-Fernández e Pedro García-Teodoro. Survey and taxonomy of botnet research through life-cycle. *ACM Comput. Surv.*, 45(4):45:1–45:33. ISSN 0360-0300. doi: 10.1145/2501654.2501659. URL http://doi.acm.org/10.1145/2501654.2501659. Citado na pág. 1, 8, 9, 11, 12

RouteFlow (2014) RouteFlow. Routeflow, 2014. URL https://sites.google.com/site/routeflow/. Acessado em 15/06/2014. Disponível em https://sites.google.com/site/routeflow/. Citado na pág. 45

Russell (2011) Matthew A. Russell. *Mining the Social Web - Analyzing Data from Facebook, Twitter, LinkedIn, and Other Social Media Sites*. O'Reilly. ISBN 978-1-449-38834-8. Citado na pág. 87

Russell e Norvig (2003) Stuart J. Russell e Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 ed. ISBN 0137903952. Citado na pág. xi, 25, 26

Ryu (2014) Ryu. Ryu sdn framework, 2014. URL http://osrg.github.io/ryu/. Acessado em 15/06/2014. Disponível em http://osrg.github.io/ryu/. Citado na pág. 45

Sabahi e Movaghar (2008) F. Sabahi e A. Movaghar. Intrusion detection: A survey. Em *Systems and Networks Communications, 2008. ICSNC '08. 3rd International Conference on*, páginas 23–26. doi: 10.1109/ICSNC.2008.44. Citado na pág. 15, 16

Sanok (2005) Daniel J. Sanok, Jr. An analysis of how antivirus methodologies are utilized in protecting computers from malicious code. Em *Proceedings of the 2Nd Annual Conference on Information Security Curriculum Development*, InfoSecCD '05, páginas 142–144, New York, NY, USA. ACM. ISBN 1-59593-261-5. doi: 10.1145/1107622.1107655. URL http://doi.acm.org/10.1145/1107622.1107655. Citado na pág. 8, 16

SANS (2016) SANS. Denial of service attacks and mitigation techniques: Real time implementation with detailed analysis, 2016. URL https://www.sans.org/. Acessado em 25/02/2016. Disponível em https://www.sans.org/reading-room/whitepapers/detection/denial-service-attacks-mitigation-techniques-real-time-implementation-detailed-analysi-33764. Citado na pág. 101

Santos *et al.* (2012) L. A. F. Santos, R. Campiolo, M. A. Gerosa e D. M. Batista. Análise de mensagens de segurança postadas no twitter. Em *Anais do IX SBSC*, páginas 791–804. Citado na pág. 59, 87, 89, 90, 93, 135, 136

Santos *et al.* (2013) L. A. F. Santos, R. Campiolo, M. A. Gerosa e D. M. Batista. Detecção de alertas de segurança em redes de computadores usando redes sociais. Em *Anais do XXXI SBRC*, páginas 791–804. Citado na pág. xii, 59, 87, 88, 89, 90, 91, 93, 94, 95, 96, 129, 135, 136

Santos *et al.* **(2014)** Luiz Arthur F. Santos, Rodrigo Campiolo e Daniel Macêdo Batista. Uma arquitetura autonômica para detecção e reação a ameaças de segurança em redes de computadores. Em *Anais do 4o Workshop em Sistemas Distribuídos Autonômicos, SBRC 2014*, páginas 45–48. Citado na pág. 57, 130, 139, 140, 145

Santos *et al.* **(2016)** Luiz Arthur F. Santos, Rodrigo Campiolo, Wagner Monteverde e Daniel Macêdo Batista. Abordagem autonômica para mitigar ciberataques em lans. Em *SBRC 2016*, Salvador, Bahia. URL http://XXXXX/152417.pdf. Citado na pág. 57, 139, 140, 146

Scarfo (2012) Antonio Scarfo. New security perspectives around byod. Em *Proceedings of the 2012 Seventh International Conference on Broadband, Wireless Computing, Communication and Applications*, BWCCA '12, páginas 446–451, Washington, DC, USA. IEEE Computer Society. ISBN 978-0-7695-4842-5. doi: 10.1109/BWCCA.2012.79. URL http://dx.doi.org/10.1109/BWCCA.2012.79. Citado na pág. 20

Shen *et al.* **(2009)** Liuqing Shen, Jindong Wang, Kun Wang e Hengwei Zhang. The design of intelligent security defensive software based on autonomic computing. Em *Intelligent Computation Technology and Automation, 2009. ICICTA '09. Second International Conference on*, volume 1, páginas 489–491. doi: 10.1109/ICICTA.2009.125. Citado na pág. 49

Shi *et al.* **(2005)** Weidong Shi, H.-H.S. Lee, Guofei Gu, Laura Falk, T.N. Mudge e M. Ghosh. An intrusion-tolerant and self-recoverable network service system using a security enhanced chip multiprocessor. Em *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, páginas 263–273. doi: 10.1109/ICAC.2005.8. Citado na pág. 53

Shin *et al.* **(2013a)** Seungwon Shin, Phil Porras, Vinod Yegneswaran, Martin Fong, Guofei Gu e Mabry Tyson. Fresco: Modular composable security services for software-defined networks. Em *Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS'13)*. Citado na pág. 150

Shin *et al.* **(2013b)** Seungwon Shin, Vinod Yegneswaran, Phillip Porras e Guofei Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. Em *Proceedings of the 2013 ACM SIGSAC CCS '13*, páginas 413–424. ISBN 978-1-4503-2477-9. doi: 10.1145/2508859.2516684. URL http://doi.acm.org/10.1145/2508859.2516684. Citado na pág. 144, 150

Sibai e Menasce (2011) Faisal M. Sibai e D. Menasce. Defeating the insider threat via autonomic network capabilities. Em *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on*, páginas 1–10. doi: 10.1109/COMSNETS.2011.5716431. Citado na pág. 50

Sibai e Menasce (2012) Faisal M. Sibai e D. Menasce. Countering network-centric insider threats through self-protective autonomic rule generation. Em *Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference on*, páginas 273–282. doi: 10.1109/SERE.2012.40. Citado na pág. 50

Smith e Nettles (2004) J.M. Smith e S.M. Nettles. Active networking: one view of the past, present, and future. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 34(1):4–18. ISSN 1094-6977. doi: 10.1109/TSMCC.2003.818493. Citado na pág. 31

SNAC (2014) SNAC. Snac - openflowhub, 2014. URL http://www.openflowhub.org/display/Snac/SNAC+Home. Acessado em 15/06/2014. Disponível em http://www.openflowhub.org/display/Snac/SNAC+Home. Citado na pág. 45

Somayaji *et al.* **(2008)** Anil Somayaji, Michael Locasto e Jan Feyereisl. The future of biologically-inspired security: is there anything left to learn? Em *Proceedings of the 2007 Workshop on New*

Security Paradigms, NSPW '07, páginas 49–54, New York, NY, USA. ACM. ISBN 978-1-60558-080-7. doi: 10.1145/1600176.1600185. URL http://doi.acm.org/10.1145/1600176.1600185. Citado na pág. 49

Sophos (2014a) Sophos. Security threat report 2014, 2014a. URL http://www.sophos.com/en-us/medialibrary/PDFs/other/sophos-security-threat-report-2014.pdf. Acessado em 30/06/2014. Disponível em http://www.sophos.com/. Citado na pág. 1, 13, 18

Sophos (2014b) Sophos. Exploring the blackhole exploit kit, 2014b. URL http://nakedsecurity.sophos.com/exploring-the-blackhole-exploit-kit/#Contents. Acessado em 07/07/2014. Disponível em http://nakedsecurity.sophos.com/. Citado na pág. 19

Sourcefire (2013) Sourcefire. Snort, 2013. URL http://www.snort.org. Acessado em 17/11/2013. Disponível em http://www.snort.org. Citado na pág. 62

SRI e ATM (2013) SRI e ATM. Openflowsec.org, 2013. URL http://www.openflowsec.org. Acessado em 17/11/2013. Disponível em http://www.openflowsec.org. Citado na pág. 3

Srivastava e Sahu (2004) P.K. Srivastava e S. Sahu. Secured remote tracking of critical autonomic computing applications. Em *E-Tech 2004*, páginas 17–22. doi: 10.1109/ETECH.2004.1353838. Citado na pág. 48

Stallings e Brown (2014) William. Stallings e Lawrie Brown. *Segurança de Computadores - Princípios e Práticas*. Elsevier, 2 ed. ISBN 8535264493. Citado na pág. 5, 6, 13, 14, 15

Starr e Yan (2013) Barbara Starr e Holly Yan. Man behind nsa leaks says he did it to safeguard privacy, liberty, 2013. URL http://www.cnn.com/2013/06/10/politics/edward-snowden-profile/. Acessado em 08/07/2014. Disponível em http://www.cnn.com/. Citado na pág. 18, 20

Sterritt *et al.* (2005) Roy Sterritt, Manish Parashar, Huaglory Tianfield e Rainer Unland. A concise introduction to autonomic computing. *Adv. Eng. Inform.*, 19(3):181–187. ISSN 1474-0346. doi: 10.1016/j.aei.2005.05.012. URL http://dx.doi.org/10.1016/j.aei.2005.05.012. Citado na pág. 22

Suh *et al.* (2010) Junho Suh, Hoon-gyu Choi, Wonjun Yoon, Taewan You, Ted Kwon e Yanghee Choi. Implementation of content-oriented networking architecture (cona): A focus on ddos countermeasure. Em *Proceedings of 1st European NetFPGA Developers Workshop*. Citado na pág. 151

Symantec (2013) Symantec. Internet security threat report 2013. Relatório Técnico 18, Symantec Corporation. Acessado em 28/11/2013. Disponível em http://www.symantec.com/. Citado na pág. 1

Symantec (2014) Symantec. Istr - internet security threat report appendix 2014, 2014. Acessado em 30/06/2014. Disponível em http://www.symantec.com/. Citado na pág. 12, 17, 18, 19

Symantec (2015) Symantec. Internet security threat report 2015. Relatório Técnico 20, Symantec. Acessado em 01/04/2015. Disponível em http://www.symantec.com/. Citado na pág. 1, 18, 19

Talib *et al.* (2010) S. Talib, N.L. Clarke e S.M. Furnell. An analysis of information security awareness within home and work environments. Em *Availability, Reliability, and Security, 2010. ARES '10 International Conference on*, páginas 196–203. doi: 10.1109/ARES.2010.27. Citado na pág. 5, 8, 12

Tanenbaum (2007) Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd ed. ISBN 9780136006633. Citado na pág. 5, 8, 44, 45, 74, 77

Tanenbaum e Wetherall (2010) Andrew S. Tanenbaum e David J. Wetherall. *Computer Networks*. Prentice Hall Press, Upper Saddle River, NJ, USA, 5th ed. ISBN 0132126958, 9780132126953. Citado na pág. 29, 37

Tanenbaum e Wetherall (2013) Andrew S. Tanenbaum e David J. Wetherall. *Computer Networks*. Pearson Education. ISBN 1292024224, 9781292024226. Citado na pág. 2, 5, 10, 14, 15, 30, 66, 74, 80, 82, 90, 110

Teles *et al.* (2011) Ariel Soares. Teles, Francisco J. S. Silva e Zair Abdelouahab. Computação autonômica aplicada a segurança de redes. Em SBC, editor, *ERCEMAPI 2011 - Livro Texto*, páginas 54–78. URL http://www.die.ufpi.br/ercemapi2011/minicursos/MC1.pdf. Citado na pág. 25, 26, 27

Trustware (2014) Trustware. 2014 trustware global security report. Relatório técnico. URL http://www2.trustwave.com/rs/trustwave/images/2014_Trustwave_Global_Security_Report.pdf. Acessado em 27/06/2014. Disponível em http://www.trustwave.com/. Citado na pág. 1, 12, 13, 19

Trustware (2016) Trustware. 2016 trustware global security report. Relatório técnico. Acessado em 14/07/2016. Disponível em http://www.trustwave.com/. Citado na pág. 1, 18, 19

Twitter4J (2016) Twitter4J. Twitter4j - a java library for the twitter api, 2016. Acessado em 28/01/2016. Disponível em http://twitter4j.org/en/index.html. Citado na pág. 89

Varas e Hirsch (2009) C. Varas e T. Hirsch. Self protection through collaboration using d-caf: A distributed context-aware firewall. Em *Emerging Security Information, Systems and Technologies, 2009. SECURWARE '09. Third International Conference on*, páginas 179–184. doi: 10.1109/SECURWARE.2009.35. Citado na pág. 51

Wailly *et al.* (2012) Aurélien Wailly, Marc Lacoste e Hervé Debar. Vespa: multi-layered self-protection for cloud resources. Em *Proceedings of the 9th international conference on Autonomic computing*, ICAC '12, páginas 155–160, New York, NY, USA. ACM. ISBN 978-1-4503-1520-3. doi: 10.1145/2371536.2371564. URL http://doi.acm.org/10.1145/2371536.2371564. Citado na pág. 50

Wan e Alagar (2006) Kai Yu Wan e V. Alagar. Security contexts in autonomic systems. Em *Computational Intelligence and Security, 2006 International Conference on*, volume 2, páginas 1523–1527. doi: 10.1109/ICCIAS.2006.295315. Citado na pág. 49

Wang e Ma (2007) Yi-Min Wang e Ming Ma. Strider search ranger: Towards an autonomic anti-spam search engine. Em *Autonomic Computing, 2007. ICAC '07. Fourth International Conference on*, páginas 32–32. doi: 10.1109/ICAC.2007.38. Citado na pág. 49

Wget (2016) Wget. Gnu wget, 2016. Acessado em 25/04/2016. Disponível em http://www.gnu.org/software/wget/. Citado na pág. 104

White *et al.* (2004) S.R. White, J.E. Hanson, I. Whalley, D.M. Chess e J.O. Kephart. An architectural approach to autonomic computing. Em *Autonomic Computing, 2004. Proceedings. International Conference on*, páginas 2–9. doi: 10.1109/ICAC.2004.1301340. Citado na pág. 27

Winkler (2011) J. R. Vic Winkler. *Securing the Cloud: Cloud Computer Security Techniques and Tactics*. Syngress Publishing. ISBN 1597495921, 9781597495929. Citado na pág. 20

Wooldridge e Jennings (1994) M. J. Wooldridge e N. R. Jennings. Agent theories, architectures and languages: A survey. Em M. J. Woolridge e N. R. Jennings, editors, *ECAI94 Workshop on Agent Theories Architectures and Languages*, páginas 1–32. URL http://eprints.soton.ac.uk/252177/. Citado na pág. 22

Xing *et al.* (2013) Tianyi Xing, Dijiang Huang, Le Xu, Chun-Jen Chung e P. Khatkar. Snortflow: A openflow-based intrusion prevention system in cloud environment. Em *Research and Educational Experiment Workshop (GREE), 2013 Second GENI*, páginas 89–92. doi: 10.1109/GREE.2013.25. Citado na pág. 3, 150

Yau *et al.* **(2006)** Stephen S. Yau, Yisheng Yao e Min Yan. Development and runtime support for situation-aware security in autonomic computing. Em *Proceedings of the Third international conference on Autonomic and Trusted Computing*, ATC'06, páginas 173–182, Berlin, Heidelberg. Springer-Verlag. ISBN 3-540-38619-X, 978-3-540-38619-3. doi: 10.1007/11839569_17. URL http://dx.doi.org/10.1007/11839569_17. Citado na pág. 48

Yuan *et al.* **(2014)** Eric Yuan, Naeem Esfahani e Sam Malek. A systematic survey of self-protecting software systems. *ACM Trans. Auton. Adapt. Syst.*, 8(4):17:1–17:41. ISSN 1556-4665. doi: 10.1145/2555611. URL http://doi.acm.org/10.1145/2555611. Citado na pág. 3

Zander e Forchheimer (1988) J. Zander e R. Forchheimer. The softnet project: a retrospect. Em *Electrotechnics, 1988. Conference Proceedings on Area Communication, EUROCON 88., 8th European Conference on*, páginas 343–345. doi: 10.1109/EURCON.1988.11172. Citado na pág. 31

Zargar *et al.* **(2013)** S.T. Zargar, J. Joshi e D. Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *Communications Surveys Tutorials, IEEE*, 15(4):2046–2069. ISSN 1553-877X. doi: 10.1109/SURV.2013.031413.00127. Citado na pág. 9, 10, 15

Zhang e Shen (2009) Zonghua Zhang e Hong Shen. M-aid: An adaptive middleware built upon anomaly detectors for intrusion detection and rational response. *ACM Trans. Auton. Adapt. Syst.*, 4(4):24:1–24:35. ISSN 1556-4665. doi: 10.1145/1636665.1636670. URL http://doi.acm.org/10.1145/1636665.1636670. Citado na pág. 51

Zhou e Foley (2004) Hongbin Zhou e Simon N. Foley. A collaborative approach to autonomic security protocols. Em *Proceedings of the 2004 workshop on New security paradigms*, NSPW '04, páginas 13–21, New York, NY, USA. ACM. ISBN 1-59593-076-0. doi: 10.1145/1065907.1066029. URL http://doi.acm.org/10.1145/1065907.1066029. Citado na pág. 48