UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE ENGENHARIA
CAMPUS DE ILHA SOLTEIRA

RICARDO CESAR CÂMARA FERRARI

REDE DEFINIDA POR *SOFTWARE* PARA A DETECÇÃO DE ANOMALIAS E CONTRAMEDIDAS DE SEGURANÇA EM *SMART GRID*

Ilha Solteira
2018

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Ilha Solteira

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

RICARDO CESAR CÂMARA FERRARI

REDE DEFINIDA POR *SOFTWARE* PARA A DETECÇÃO DE ANOMALIAS E CONTRAMEDIDAS DE SEGURANÇA EM *SMART GRID*

Tese apresentada à Faculdade de Engenharia de Ilha Solteira – UNESP como parte dos requisitos para obtenção do título de doutor. Especialidade: Automação.

Prof. Dr. Ailton Akira Shinoda
Orientador

Ilha Solteira
2018

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

F375r

Ferrari, Ricardo Cesar Câmara.
Rede definida por software para a detecção de anomalias e contramedidas de segurança em Smart Grid / Ricardo Cesar Câmara Ferrari. -- Ilha Solteira: [s.n.], 2018
102 f. : il.

Tese (doutorado) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira. Área de conhecimento: Automação, 2018

Orientador: Ailton Akira Shinoda
Co-orientador: Christiane Marie Schweitzer
Inclui bibliografia

1. Smart Grid. 2. SDN. 3. Segurança. 4. DDoS. 5. POX.

João Josué Barbosa
Serviço Técnico de Biblioteca e Documentação
Diretor Técnico
CRB 8-5642

unesp

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Ilha Solteira

CERTIFICADO DE APROVAÇÃO

TÍTULO DA TESE: REDE DEFINIDA POR *SOFTWARE* PARA A DETECÇÃO DE ANOMALIAS E CONTRAMEDIDAS DE SEGURANÇA EM *SMART GRID*

AUTOR: RICARDO CESAR CAMARA FERRARI
ORIENTADOR: AILTON AKIRA SHINODA

Aprovado como parte das exigências para obtenção do Título de Doutor em ENGENHARIA ELÉTRICA, área: AUTOMAÇÃO pela Comissão Examinadora:

Prof. Dr. AILTON AKIRA SHINODA
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Profa. Dra. BERENICE CAMARGO DAMASCENO
Departamento de Matemática / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. CARLOS ANTONIO ALVES
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Profa. Dra. CAROLINA GOULART DE CARVALHO
FIU / Faculdades Integradas Urubupungá

Prof. Dr. LEANDRO APARECIDO VILLAS
Departamento de Sistemas de Computação / Universidade Estadual de Campinas - UNICAMP

Ilha Solteira, 01 de março de 2018

DEDICATÓRIA

Dedico, primeiramente, esta tese, a Deus, aos meus queridos pais, Antônio Ferrari e Terezinha Câmara Ferrari, à minha esposa Érica Rejane Minatel Ferrari, a todos que, de alguma forma, contribuíram para mais essa conquista, em especial para minha irmã Rosicler Câmara Ferrari Marinho, que não está mais entre nós fisicamente, mas que, com certeza, me acompanha espiritualmente.

AGRADECIMENTOS

A Deus, por me dar saúde e força para vencer mais esta etapa da minha vida. Aos meus pais Antônio Ferrari, Terezinha Câmara Ferrari e à minha esposa Érica Rejane Minatel Ferrari, pela paciência nos momentos mais críticos, minhas irmãs Rosicler, Janete e Raquel, sempre presentes nos momentos mais difíceis, dando todo o apoio necessário, com muita paciência e carinho.

Aos Professores doutores Ailton Akira Shinoda e Christiane Marie Schweitzer, pela compreensão, paciência e competência e, relação à orientação e à elaboração deste trabalho.

Aos amigos de pós-graduação que me ajudaram sempre que precisei, dando sugestões e materiais de apoio, sempre muito úteis.

À banca examinadora que analisará o conteúdo deste trabalho contribuindo com sugestões.

“Aquilo que está escrito no coração não necessita de agendas porque a gente não esquece. O que a memória ama fica eterno”. Rubem Alves.

RESUMO

O trabalho propõe uma aplicação com o uso de desvio padrão para definir limites máximos e mínimos de pacotes e *bytes* para detecção de anomalias nos fluxos de comunicação entre mestre e escravos com o uso do protocolo DNP3 (*Distributed Network Protocol v3.0*) em uma *Smart Grid*, além de detecção e bloqueio de ataques originados de máquinas intrusas ou conhecidas. Atualmente, diversas pesquisas vêm sendo desenvolvidas sobre uso de sistemas *Smart Grid*, no entanto, sua implantação possui alguns fatores de risco. Esses fatores estão associados às redes de transmissão de dados, às tecnologias de aquisição e controle das informações, e às vulnerabilidades intrínsecas da união dessas tecnologias. A principal motivação dessa proposta origina-se da necessidade de se garantir segurança dos sistemas *Smart Grid* e o potencial apresentado pelas Redes Definidas por *Software* (*Software Defined Networking* – SDN) em analisar os fluxos de dados em um *switch*. Assim, a investigação dessas vulnerabilidades, bem como, a identificação de situações de ataques são relevantes a fim de propor soluções de defesa. Para isto, a tecnologia de SDN apresentou-se como uma solução viável e otimizada para a proteção de sistemas *Smart Grid*, permitindo que seja realizado um monitoramento dos fluxos entre mestre e escravos, e a coleta de informações para análise, abrindo oportunidades para aplicações de segurança em *Smart Grid*. Dessa forma, foram realizados três experimentos, o primeiro com o objetivo de mostrar a vulnerabilidade de uma *Smart Grid*, o segundo com o intuito de analisar uma aplicação SDN em uma *Smart Grid* e o terceiro com dois ataques DDoS (*Distributed Denial of Service*) em uma *Smart Grid.* O primeiro ataque a partir de máquinas intrusas e o segundo ataque, de escravas, permitindo analisar e monitorar o fluxo de dados e o bloqueio das portas em um *Open vSwitch* (OVS). Nesse contexto, um componente de um controlador SDN foi modificado para adicionar segurança e monitoramento da rede, tendo um comportamento satisfatório, identificando anomalias e conseguindo realizar bloqueios de portas das máquinas atacantes.

Palavras-chave: *Smart Grid*. SDN. Segurança. DDoS. POX.

ABSTRACT

The work proposes an application with the use of standard deviation to define limits of maximum and minimum of packets and bytes for detection of anomalies in the communication flows between master and slave using the Distributed Network Protocol v3.0 (DNP3), besides the detection and blocking of attacks originated from intrusive or known machines. Currently several researches have been developed on the use of Smart Grid systems, however, its implementation has some risk factors. These factors are associated with data transmission networks, information acquisition and control technologies and intrinsic vulnerabilities of the union of these technologies. The main motivation of this proposal comes from the need to guarantee security of Smart Grid systems and the potential presented by Software Defined Networking (SDN). Thus, the investigation of these vulnerabilities, as well as, identification of situations of attacks are relevant in order to propose defense solutions. For this, the SDN technology has presented a viable and optimized solution for the protection of Smart Grid systems, allowing the monitoring of master-slave flows and the collection of information for analysis, opening opportunities for security applications in Smart Grid. In this way, three experiments were carried out, the first to show the vulnerability of an insecure Smart Grid, the second to analyze a SDN application in a Smart Grid and the third with two Distributed Denial of Service (DDoS) attack on a Smart Grid, the first from intrusive machines and the second from slaves, allowing to analyze and monitor the data flow and the lock of the doors in an Open vSwitch (OVS). In this context, a component of an SDN controller has been modified to add security and monitoring of the network, having a satisfactory behavior, identifying anomalies and being able to perform port blocking of the attacking machines

Keywords: *Smart Grid*. SDN. Segurança. DDoS. POX**.**

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

AMI	*Advanced Metering Infrastructure*
API	*Application program interface*
ATM	*Asynchronous Transfer Mode*
BAN	*Building Area Network*
CPU	*Central Processing Unit*
CSV	*Comma Separated Values*
DDoS	*Distributed Denial of Service*
Dn	Desvio n
DNP	*Distributed Network Protocol*
DNP3	*Distributed Network Protocol v3.0*
DoS	*Denial of Service*
DSM	*Demand side management*
EISA	*Energy Independence and Security Act*
FDDI	*Fiber Distributed Data Interface*
GB	*Giga Bytes*
Geneve	*Generic Network Virtualization Encapsulation*
GENI	*Global Environment for Network Innovations*
GPL 2	*GNU General Public License 2*
GRE	*Generic Routing Encapsulation*
HAN	*Home Area Network*
HDLC	*High-Level Data Link Control*
HW	*Hardware*
IAN	*Industrial Area Network*
ICMP	*Internet Control Message Protocol*
IDE	*Integrated Drive Electronics*
IDS	*Intrusion detection system*
IEC	*International Electrotechnical Commission*
IEDs	*Intelligent Electronic Devices*
IEEE	*Institute of Electrical and Electronics Engineers*
IGMP	*Internet Group Management Protocol*
IoT	*Internet of Things*
IP	*Internet Protocol*

IPS	*Intrusion Prevention System*
IPSec	*Internet Protocol Security*
IPv4	*Internet Protocol version 4*
KVM	*Kernel-based Virtual Machine*
LAN	*Local Area Network*
LISP	*Locator/Identifier Separation Protocol*
LTS	*Long Term Support*
MV1	Máquina Virtual 1
MV2	Máquina Virtual 2
MV3	Máquina Virtual 3
NCSA	*National Center for Supercomputing Applications*
OFP	*OpenFlow Protocol*
ONF	*Open Networking Foundation*
OPC	*Object Linking and Embedding (OLE) for process control OPC*
OVS	*Open vSwitch*
PC	*Personal Computer*
PCAP	*Packet CAPture*
PPP	*Point-to-Point Protocol*
QoS	*Quality of Service*
RPMs	*Red Hat Package Manager*
RTT	*Round Trip Time*
RTU	*Remote Terminal Unit*
SCADA	*Supervisory Control and Data Acquisition System*
SDN	*Software Defined Network*
SNMP	*Simple Network Management Protocol*
SSL	*Secure Socket Layer*
STT	*Stateless Transport Tunneling*
SW	*SoftWare*
TCP	*Transmission Control Protocol*
TCP/IP	*Transmission Control Protocol over Internet Protocol*
TLS	*Transport Layer Security*
TS	*Test Set*
UDP	*User Datagram Protocol*
USB	*Universal Serial Bus*

VM	*Virtual Machine*
VoIP	*Voice over Internet Protocol*
VXLAN	*Virtual Extensible LAN*
WEP	*Wired Equivalent Privacy*
WPA	*Wi-Fi Protected Access*
WPA2	*Wi-Fi Protected Access version 2*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

O sistema de energia elétrica convencional foi projetado para operar em uma estrutura consistindo em geração, transmissão e distribuição e com suporte para manter a confiabilidade, estabilidade e eficiência. No entanto, as concessionárias de energia, atualmente, estão enfrentando novos desafios, com a inclusão de recursos energéticos renováveis no sistema convencional, rápidas mudanças tecnológicas e diferentes tipos de dispositivos e usuários finais (MOMOH, 2012).

Com o objetivo de aumentar a integração e um uso de energia mais inteligente, com recursos energéticos distribuídos, a *Smart Grid* foi proposta para substituir o sistema de energia convencional, possibilitand16o um sistema inteligente de energia e comunicação de dados (KIM; FILALI; KO; 2015).

A *Smart Grid* é uma extensa combinação de redes de energia, redes de comunicação e sistemas de gerenciamento de informações, contribuindo para a geração de energia verde (gerada a partir de recursos renováveis) e econômica. Uma *Smart Grid* pode incluir uma infraestrutura de comunicação de dados integrada com uma rede elétrica que coleta dados sobre transmissão, distribuição e consumo de energia, em tempo real. Assim, uma *Smart Grid* é a convergência contínua de sistemas de rede elétricas existentes e tecnologias da informação e comunicação (KIM; FILALI; KO; 2015).

Quando uma nova tecnologia surge, novos problemas, também, aparecem, principalmente na área de segurança. Como apresentado por Momoh (2012), a EISA (*Energy Independence and Security Act*) de 2007, lei promulgada pelo governo norte americano, mostra uma série de *Smart Grids* que podem realizar previsões, adaptar-se e reconfigurar-se de forma eficiente e confiável. O objetivo da modernização é manter uma infraestrutura de eletricidade confiável e segura, que atenda o crescimento da demanda futura. Em resposta à legislação, a comunidade de pesquisa e educação dos EUA está ativamente envolvida, no que se refere a *Smart Grid*, em:

- Programa de pesquisa e desenvolvimento;
- Desenvolvimento de padrões e proteção amplamente aceitos;
- Desenvolvimento de infraestrutura para permitir a implantação;
- Certeza de confiabilidade e segurança do sistema;
- Política e motivação para incentivar a tecnologia para geração, transmissão e distribuição.

Com a preocupação nas novas vulnerabilidades na segurança em *Smart Grid,* Zhang et al. (2013), Dong et al. (2015) e Qin et al. (2014) avaliam a utilização de um novo modelo para auxiliar em seu gerenciamento, chamado de SDN (*Software Defined Network*). SDN vem despertando muita atenção por ser um novo conceito de arquitetura de rede que abstrai as funcionalidades de controle do *hardware* para um controlador de *software* externo (plano de controle). Isto insere a SDN como uma opção conveniente, possibilitando centralizar o plano de controle da rede em uma única máquina. Em virtude de o controlador ser implementado como *software*, permite o controle do fluxo nos comutadores de rede, proporcionando qualquer tipo de configuração ou autoconfiguração mais ágil. Dessa forma, uma SDN é vista como uma opção interessante de utilização na comunicação de uma rede *Smart Grid*, pois os pacotes DNP3 podem ser facilmente capturados e replicados para a realização de ataques, podendo a SDN ser aplicada na segurança das comunicações, como monitoramento, bloqueios de portas, identificação de anomalias, desvios de rotas, qualidade de serviço e outros.

1.1 OBJETIVOS DO TRABALHO

Com base em uma revisão bibliográfica realizada, várias pesquisas abordam a utilização de SDN para fornecer recursos para redes *Smart Grid*, porém, não foi encontrado nenhum estudo que trate do monitoramento no fluxo de uma *Smart Grid* para detecção de anomalias e contramedidas para possíveis tentativas de ataques ou mau funcionamento da rede no fluxo de dados de um *switch* com suporte SDN ou em escravos individuais, levando em consideração o cálculo de desvio padrão em diferentes intervalos.

Motivada pelas oportunidades que uma SDN pode proporcionar para uma *Smart Grid* em relação a sistemas de segurança e aplicação de outras técnicas para detecção de intrusos, a tese apresenta um experimento em que mestre e escravo são atacados a fim de demonstrar a vulnerabilidade em uma rede *Smart Grid* insegura. Esse experimento tem o objetivo de analisar quais os comportamentos que mestre e escravo podem ter sob ataques de replicação de pacotes, alterando a velocidade que os pacotes são replicados e o tempo de ataque ininterrupto.

Um segundo experimento simula a troca de dados entre dispositivos de uma rede *Smart Grid*, por meio de um *switch* com recursos *OpenFlow* ligado a um controlador, a fim de identificar se o aumento de nós, em uma *Smart Grid,* pode ocasionar um gargalo entre o comutador e o controlador, a ponto de

inviabilizar a aplicação de SDN em ambientes críticos, com o propósito de analisar e medir o RTT do fluxo de dados entre o OVS (*Open vSwitch*) e o controlador SDN.

Tendo em vista que uma comunicação rápida entre o *switch* e o controlador é fundamental para um bom monitoramento da rede *Smart Grid,* a avaliação é realizada com o aumento de máquinas na rede para identificar se, com um maior número de escravos, o fluxo pode ser afetado ou provocar um gargalo na rede.

Por fim, um terceiro experimento propõe uma aplicação capaz de encontrar todos os nós em uma determinada rede e definir o conjunto de escravos, passando a ter conhecimento de todas as máquinas da rede. Após o reconhecimento, inicia-se o cálculo dos desvios padrões para diferentes momentos estabelecidos pelo administrador da rede, em um ambiente para que, posteriormente sejam possíveis analisar eventuais anomalias nos fluxos em diferentes momentos do monitoramento, tanto para fluxos individuais como para o fluxo total do *switch*. O controlador pode definir valores por meio de cálculos em um momento de aquisição de dados e, posteriormente, com esses valores, identificar anomalias no fluxo, detectar e bloquear nós, que não fazem parte da rede pré-definida ou que sejam nós conhecidos da rede, mas iniciem algum tipo de ataque.

1.2 ORGANIZAÇÃO DO TRABALHO

Esta tese apresenta informações sobre a segurança em *Smart Grid,* apresentando primeiramente como o comportamento de um mestre ou escravo sob ataque de repetição de pacotes pode influenciar a comunicação entre eles, em seguida analisar o desempenho entre o *switch OpenFlow* e o controlador SDN, que tem o objetivo de auxiliar a *Smart Grid* identificando todos os escravos da rede, por fim, testar a aplicação desenvolvida que monitora anomalias em cada fluxo entre mestre e escravo e, contramedidas de segurança, como bloqueio de portas, caso seja considerado um ataque. O trabalho está organizado em capítulos, com uma breve descrição dos mesmos a seguir:

- **Capítulo 2:** ilustra o estado da arte sobre *Smart Grid* e Redes Definidas por Software, mostrando que pesquisas estão sendo realizadas para aplicações

com as duas tecnologias em diversas áreas, como roteamento, segurança e monitoramento;

- **Capítulo 3:** apresenta uma breve descrição sobre *Smart Grid*, mostrando sua arquitetura básica, o funcionamento, aplicações como o sistema *SCADA* (*Supervisory Control and Data Acquisition System*), protocolo de comunicação, as vulnerabilidades e tipos de ataques que podem ser explorados. Também apresenta uma introdução das Redes Definidas por *Software,* com seus componentes básicos, *framework* de comunicação, o controlador utilizado para os experimentos e o ambiente GENI para realização dos testes;
- **Capítulo 4:** descreve a proposta da tese, com as ferramentas utilizadas para os experimentos e principalmente os três cenários desenvolvidos no trabalho, com as ferramentas e planos de testes para cada um;
- **Capítulo 5:** analisa os resultados, apresentando os comportamentos em cada cenário;
- **Capítulo 6:** apresenta as conclusões e sugestões para trabalhos futuros.

2 TRABALHOS RELACIONADOS

No que se refere ao contexto sobre *Smart Grid* e Redes Definidas por Software, alguns trabalhos foram encontrados com o objetivo de analisar ou propor recursos a um sistema *Smart Grid* de forma que seja viável a aplicação, evitando problema de desempenho e garantindo mais segurança na rede.

Kim, Filali e Ko (2015) investigaram um novo modelo chamado *Smart Grid* habilitada para SDN. Além disso, comparam estudos recentes do modelo com base em suas funcionalidades de serviço, e apresentam mais problemas de pesquisa em relação a este. O modelo constitui uma solução de rede emergente e promissora, mas precisa ser otimizado para uso em um ambiente de *Smart Grid*, pois a sobrecarga de comunicação entre o controlador SDN e os dispositivos precisa ser otimizada para evitar o congestionamento da rede e os problemas de atraso computacional.

Com o intuito de investigar problemas de fluxo em SDN, foi realizado um estudo de desempenho entre multi-redes, propondo um controlador SDN para gerenciamento de fluxo na "*Internet* das Coisas" (QIN et al., 2014), conseguindo um melhor desempenho quando comparado com outras técnicas. Rinaldi et al. (2015) apresentam uma pesquisa sobre o RTT (*Round Trip Time*) entre nós da rede, ou seja, relativa ao plano de dados da SDN, concluindo que uma modelagem da rede pode ser necessária para um bom desempenho.

Kim, Filali e Ko (2015) concluíram que a complexidade e a heterogeneidade dos serviços de *Smart Grid* estão aumentando, rapidamente, devido ao aumento do uso de IEDs (*Intelligent Electronic Devices*), automação e integração de serviços entre domínios de utilidade.

Esse processo ainda se encontra nos estágios iniciais. Além disso, apresenta uma comparação dos atuais sistemas de *Smart Grid* habilitadas para SDN e categoriza os serviços e aplicativos de rede examinados que foram desenvolvidos com base no modelo SDN.

Na Tabela 1, 3 arquiteturas são consideradas como candidatas para soluções SDN e *Smart Grid,* no que diz respeito à segurança. Essas arquiteturas são apresentadas por Kim, Filali e Ko (2015), mais a proposta da tese:

Tabela 1 - Arquiteturas candidatas para soluções SDN e *Smart Grid*

Arquitetura	Características avaliadas								
	Conhecimento de contexto	Monitoramento de redes	Descoberta de vizinhos	Balanceamento de cargas	Qualidade de serviço	Segurança	Controlador SDN	Interface de rede	Rede
Molina, E. et al., 2-15	X	X	-	X	-	X	Floodlight	Ethernet	WAN/ NAN
Cahn, A et al., 2013	-	X	-	X	-	X	Custom	Ethernet	NAN
Dong, X et al., 2015	-	X	-	-	-	X	ns-2, Sim	Ethernet	WAN/ NAN
Aplicação proposta	-	X	X	-	-	X	POX	Ethernet	NAN/HAN

X: suportado – não suportado

Fonte: Kim, Filali e Ko (2015) modificado

A arquitetura de Molina et al. (2015) trata de sistemas baseados em IEC 61850 (*International Electrotechnical Commission*), capaz de realizar reencaminhamento de tráfego de dados, QoS (*Quality of Service*) e distribuição da carga de tráfego. Utiliza, também, o coletor de dados *sFlow* (solução proprietária) para detecção de limite excedido e comunicação com controlador. A segunda arquitetura, de Cahn et al. (2013) descreve, apenas, um protótipo e não mostra nenhuma implementação ou resultados que possam ser discutidos como comparação. Por fim, em Dong et al. (2015) apresentam recursos como: estabelecer rotas dinâmicas, restabelecer um roteamento comprometido e troca de canais com cifragem, em possíveis ataques. O IDS (*Intrusion Detection System*) está implementado fora do controlador, isso pode ocasionar maior custo de manutenção em uma atualização por serem 2 *hosts* diferentes. Também não apresenta uma forma de identificar algum intruso na rede ou bloquear o atacante. A simulação não apresenta nenhuma forma de aprendizado para que o controlador possa monitorar a rede de forma mais precisa, levando em conta que cada rede pode se comportar de uma maneira diferente.

A arquitetura proposta possibilita o monitoramento da rede com base em desvios padrões em diferentes momentos e a única com descoberta de vizinhos, por meio de uma máquina sensor, além de possuir um sistema de segurança com contramedidas com bloqueios de portas.

2.1 CONSIDERAÇÕES FINAIS

O capítulo apresenta os trabalhos relacionados com as tecnologias *Smart Grid* e SDN, juntamente à proposta do trabalho, estruturadas em uma tabela com as principais características de cada arquitetura.

Para um melhor entendimento sobre as tecnologias empregadas, os próximos capítulos definem o funcionamento e os detalhes de uma *Smart Grid* e uma SDN.

3 APORTE TEÓRICO

As duas principais tecnologias utilizadas neste trabalho, *Smart Grid* e SDN, serão introduzidas nesta seção, trazendo a visão geral de suas principais características, para que o leitor tenha condições de entender a utilização de ambas, em relação à arquitetura proposta.

3.1 *SMART GRID*

A *Smart Grid* é definida como uma rede elétrica de grande escala que pode combinar ações de usuários que atuam como geradores, consumidores ou ambos. Os usuários podem gerar energia a partir de fontes de energia renováveis, como painéis solares e geradores elétricos. Mais recentemente, a *Smart Grid* vem enfrentando vários desafios, incluindo questões ambientais, de gerenciamento e de segurança. A estrutura da *Smart Grid,* em comparação com a rede de energia convencional, será alterada para que todas as entidades possam desempenhar um papel de produtor e consumidor (FANG et al., 2012).

Isso significa que agentes poderão produzir e consumir energia elétrica de forma autônoma. Existem várias razões para que isso aconteça, que incluem queda de pressão no horário de pico para os produtores e redução de custos de energia para os consumidores (GELLINGS et al., 2011), ou seja, a produção de energia elétrica pelo consumidor poderá satisfazer suas demandas de compra da rede de serviços públicos, além de poderem vender a energia extra com lucro (LONI; PARAND, 2017).

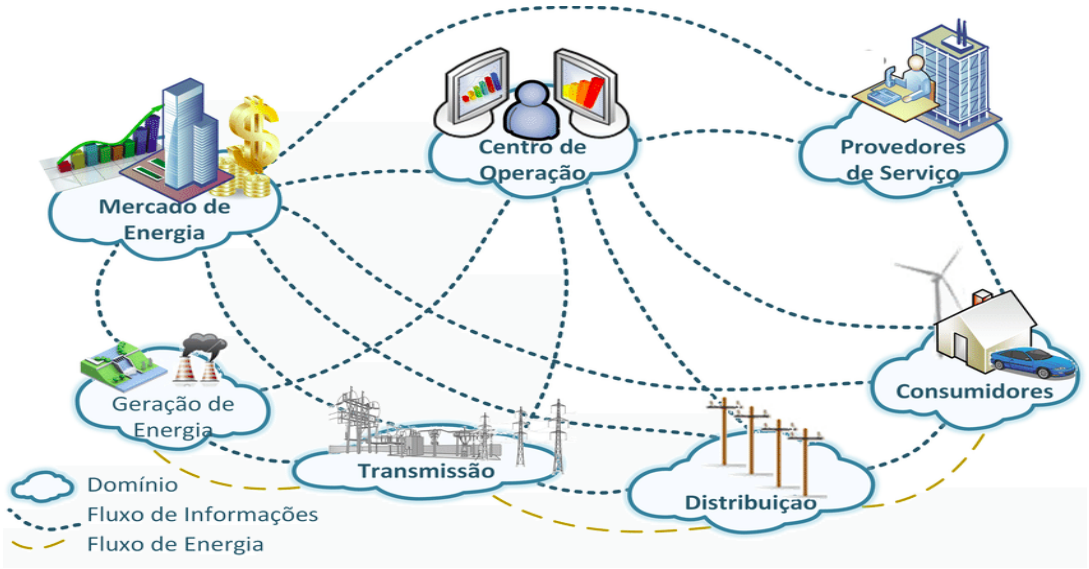
A tecnologia *Smart Grid* cria um ambiente de convergência entre as infraestruturas de geração, transmissão, distribuição e tecnologia da informação, que permite o intercâmbio de informações e ações de controle entre os vários segmentos da rede elétrica. Uma *Smart Grid*, por meio do uso de medidores inteligentes e tecnologia de comunicação digital bidirecional, implementa uma AMI (*Advanced Metering Infrastructure*) para predição de carga nas áreas residenciais e, assim, contribui para a eficiência energética. Essa comunicação bidirecional aumentará a satisfação do cliente, permitindo, por exemplo, programas efetivos de DSM (*Demand Side Management*) para influenciar o comportamento dos clientes em relação ao seu consumo de energia (GAUR et al., 2017).

Com o aumento da conscientização sobre mudanças climáticas, pesquisadores enfatizam a produção de energia por meio de fontes de energia renováveis, contribuindo

para geração distribuída e o funcionamento das *Smart Grids*. O crescimento esperado em relação à geração distribuída ocasionará problemas relacionados à qualidade de energia e operação eficiente do sistema de energia global. Sob as circunstâncias dadas, a *Smart Grid* poderá ser a solução para uma rede de energia elétrica mais confiável e eficiente com base na interação da rede convencional com microredes por meio de protocolos de comunicação e controles avançados (ABIDEEN; HASSAN, 2017).

Um objetivo importante da *Smart Grid* é aproveitar a infraestrutura de comunicação digital atual no controle dos sistemas de energia de forma mais eficiente. Além disso, conforme são disponibilizados mais dispositivos na *Internet* das Coisas (*Internet of Things* – IoT), com capacidade de medição e/ou controle, para um sistema de energia mais estável e eficiente, o papel da infraestrutura de comunicação torna-se mais importante.

Figura 1 - Modelo de referência de *Smart Grid* do NIST



Fonte: Lopes et al. (2016)

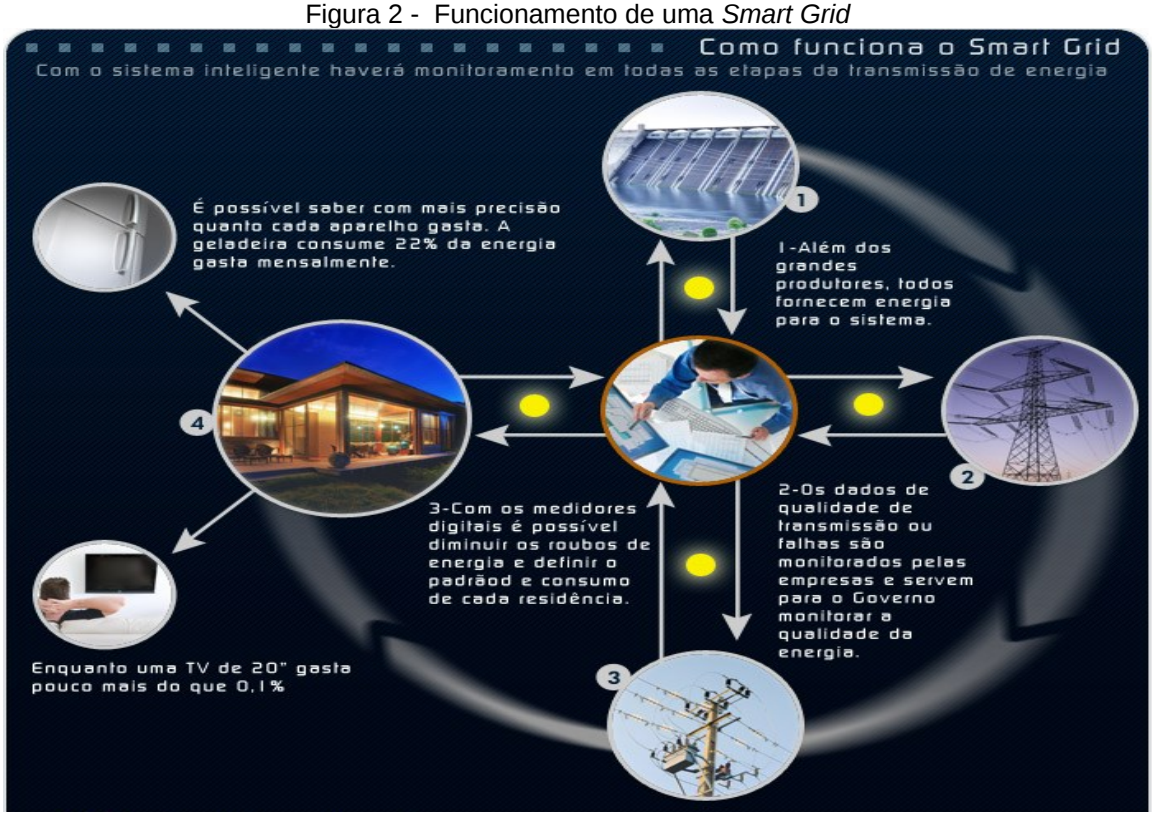
Um dos modelos de referência mais conhecidos para *Smart Grid* é proposto pelo NIST (*National Institute of Standards and Technology*), descrito por Gellings et al. (2011). Uma visão conceitual do modelo de referência de *Smart Grid* do NIST é apresentada na Figura 1.

O modelo do NIST é composto por sete domínios: geração, transmissão, distribuição, clientes, mercados, operações e prestadores de serviços. Os fluxos elétricos de duas vias estão se movendo entre os quatro domínios (geração, transmissão, distribuição e cliente), que são controlados e gerenciados pelos domínios restantes (mercado, operações e prestadores de serviços) por meio dos fluxos de comunicação.

Além disso, três clientes típicos estão na Figura 1: HAN (*Home Area Network*), BAN (*Building Area Network*) e IAN (*Industrial Area Network*), em que a infraestrutura de medição avançada (AMI) ocorre para monitorar e gerenciar a energia e a informação flui por meio de medidores inteligentes (TAN et al., 2017).

A Figura 2 ilustra, resumidamente, o funcionamento de uma *Smart Grid*, além dos elementos constituintes. A sequência de passos apresenta que, além dos grandes produtores de energia elétrica, como usinas de geração de energia solar ou hidroelétricas, os clientes, também, poderão produzir e fornecer energia para as provedoras, possibilitando ao governo e as empresas monitoramento da rede *Smart Grid* e a diminuição de fraudes e acompanhamento de consumo dos eletrodomésticos pelos clientes.

Figura 2 - Funcionamento de uma *Smart Grid*



Fonte: Camargo (2009) modificada

3.1.1 Sistemas *SCADA*

Conforme explanado por Fu e Ni (2015), o *SCADA* é um tipo de sistema automático baseado em processador com a função de supervisionar o processo de controle. O *SCADA* recebe e processa as informações relacionadas ao status de equipamentos locais ou remotos, incluindo aquisição de dados, controle e medição de

dispositivos, ajuste de parâmetros e vários sinais de alarme. O sistema *SCADA* é amplamente empregado em processos de grande escala, como sistemas de energia, transporte rodoviário, sistemas de abastecimento de água, infraestrutura, proteção ambiental, pesquisa aeroespacial entre outros. Para cumprir, de forma eficiente, a função predominante do sistema *SCADA* em um monitoramento distribuído e gerenciamento centralizado, a estrutura principal do sistema *SCADA* sempre inclui: unidades terminais remotas (*Remote Terminal Unit* - RTU), rede de comunicação e *sites* de monitoramento. Os dados de monitoramento que representam a situação real do equipamento no local são armazenados pela RTU ou enviados para os *sites* de monitoramento solicitantes. Com base nesses dados de supervisão, os *sites* de monitoramento distribuem o comando de controle para as RTUs executarem as ações correspondentes.

A comunicação entre os elementos de um sistema *SCADA* comumente é realizada por meio do protocolo DNP3, descrito por *DNP users group* (2017), que constitui um dos protocolos de comunicação de rede *SCADA* mais utilizados nesse tipo de sistema, com padrão aberto para facilitar a interoperabilidade dos fabricantes de dispositivos.

3.1.2 DNP3

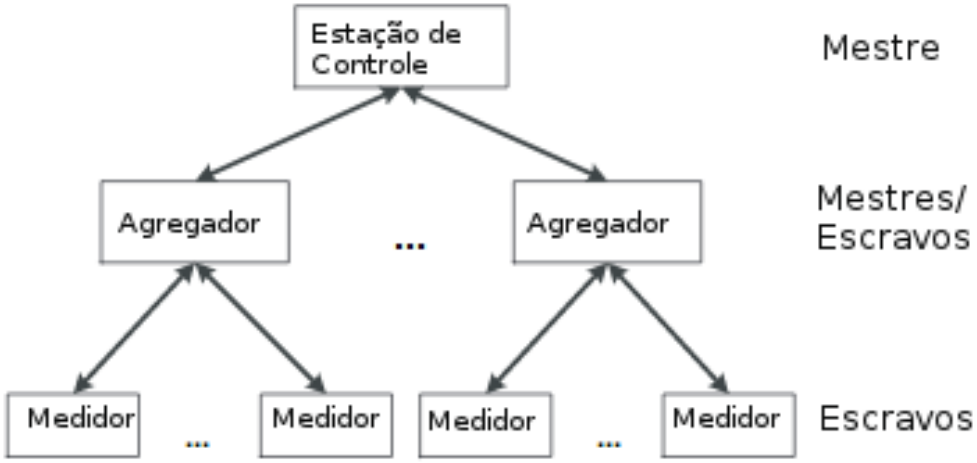
O desenvolvimento do protocolo de comunicação DNP3 constituiu um esforço abrangente para a interoperabilidade com padrões abertos entre processadores de subestação, RTUs, IEDs e estações mestres, para a indústria da eletricidade. Desde o início do DNP3, o protocolo foi utilizado em indústrias da água, transporte, petróleo e gás. É baseado nos padrões do comitê técnico da IEC 57, Grupo de Trabalho 03.

O DNP3 foi concebido para ser o mais próximo possível dos padrões que existiam no momento do desenvolvimento, com a adição de funcionalidades não identificadas na Europa, mas necessárias para aplicações norte-americanas. Ele foi desenvolvido pela *Harris*, *Distributed Automation Products*. Em novembro de 1993, a responsabilidade de definir novas especificações e a propriedade das especificações DNP3 foi transferida para o DNP3 *Users Group*, um grupo composto por utilitários e fornecedores que utilizam o protocolo.

O comitê técnico do grupo de usuários DNP3 avalia as modificações ou adições sugeridas ao protocolo e, em seguida, modifica a sua descrição, conforme indicado pelos membros do grupo de usuários. A documentação completa e abrangente do protocolo está disponível para o público. A biblioteca de documentos contém as especificações do

protocolo, bem como detalhes sobre o que é necessário nos diferentes subníveis, como implementação de autenticação segura, criação de perfis de dispositivos XML (*eXtensible Markup Language*) e procedimentos de teste de conformidade.

Figura 3 - Arquitetura *Smart Grid* de dois níveis



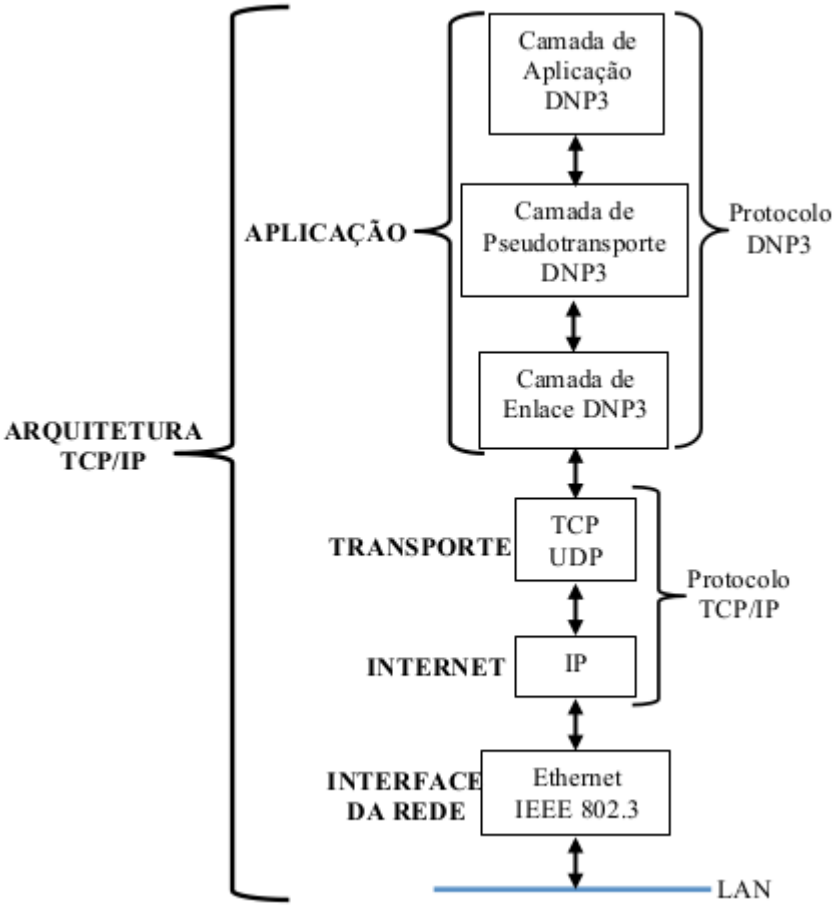
Fonte: Jin, Nicol e Yan (2011) modificado

A Figura 3 mostra a arquitetura típica de dois níveis, em que o DNP3 opera com o conceito de mestre e escravo, tendo um controle central (mestre) capaz de monitorar e gerenciar as atividade dos dispositivos eletrônicos inteligentes (escravo). Uma estação escrava pode funcionar como agregador de dados, ou seja, como um mestre DNP3 para controlar e coletar dados de dispositivos de monitoramento, sem deixar de ser um escravo DNP3 para transmitir todos os dados coletados de volta para a estação de controle central.

Com o passar dos anos, como relata Ortega et al. (2015a), houve a necessidade de expansão de comunicação com redes maiores e de maior capacidade, em que surgiu a necessidade de novas implementações do protocolo DNP3 para interagir com o protocolo TCP/IP (*Transmission Control Protocol over Internet Protocol)*.

A Figura 4 apresenta a implementação do protocolo DNP3 sobre TCP/IP, de modo que as camadas acima mencionadas não se alteram, exceto o método de sincronização, mas a mensagem é enviada de forma transparente, independentemente do TCP/IP.

Figura 4 - Implementação do protocolo DNP3 sobre TCP/IP



Fonte: Ortega (2015b) modificado

De acordo com Ortega et al. (2015a), as confirmações na camada de enlace de dados não são usadas, apenas a confirmação na camada de aplicação é empregada de forma semelhante ao que acontece na transmissão serial. A interface entre a camada de gerenciamento de conexão e a camada de transporte do protocolo TCP é implementada por meio de uma API (*Application Programming Interface*), um conjunto de rotinas de programação para acesso a um aplicativo de software. A camada de transporte é composta por dois protocolos, TCP e UDP, um dispositivo que opera DNP3 deve ter suporte para ambos os protocolos. O número da porta geralmente atribuído aos escravos é 20000.

A partir dessa condição de comunicação em redes TCP/IP, as comunicações em ambientes *Smart Grid* / *SCADA* absorveram, também, os problemas de segurança relacionados as redes TCP/IP.

3.1.3 Vulnerabilidades e segurança

Como apresentado por Hittini, Abdrabou e Zhang (2016), uma *Smart Grid* é projetada para aumentar a eficiência e a confiabilidade dos sistemas de energia.

No entanto, alcançar essa funcionalidade requer o uso de dispositivos inteligentes, o que, consequentemente, serão introduzidas muitas vulnerabilidades. A exploração de tais vulnerabilidades é muito mais crítica comparada às redes convencionais, pois um ataque bem sucedido em uma *Smart Grid* pode causar uma queda de energia em grandes áreas e, por sua vez, resultar em enormes danos, tanto para clientes como para os provedores, além do envio de informações errôneas que pode induzir em tomada de decisões inadequadas e no envio de comandos operacionais incorretos, ocasionando consequências severas. Como descrito por Zhu et al (2015), ataques de segurança têm capacidade para causar grandes apagões, além disso, manter os serviços de segurança não é uma tarefa simples, pois os ataques são de diferentes tipos, como a replicação de nó e a injeção, duplicação ou alteração de pacotes.

Além das vulnerabilidades herdadas das tecnologias TCP/IP em uma rede *Smart Grid,* Baig e Amoud (2013) apresentam a vulnerabilidade do protocolo de comunicação DNP3, sendo a principal razão para ataques em rede de controle industrial. O protocolo DNP3 é usado, principalmente, na rede de controle industrial, isolada de outras redes de computadores. Exceto os requisitos de tempo real, confiabilidade e eficiência, a segurança do protocolo não é considerada estritamente, e a organização de gerenciamento de protocolos não adicionou nenhum recurso de segurança ao protocolo DNP3.

Os problemas típicos de segurança atuais no protocolo DNP3 são os seguintes:

- Não existe uma definição relevante de autenticação;
- O protocolo DNP3 não possui mecanismo de controle de acesso;
- Falta de criptografia.

De acordo com Baig e Amoud (2013), durante o processo de comunicação do protocolo DNP3, o endereço e o comando são transmitidos em texto simples, facilitando a captura do pacote e posterior análise pelos atacantes.

Como descrevem Baig e Amoud (2013), a presença de diversos dispositivos que compõem uma *Smart Grid* acaba expondo suas vulnerabilidades, podendo ser explorada para lançar ataques maliciosos. Numerosos ataques de várias categorias podem ser

realizados contra todo o sistema *Smart Grid* ou componentes específicos deste. Para se defender contra tais ataques, são imprescindíveis identificação e detecção adequadas.

Os ataques em *Smart Grid* podem ser classificados como:

- Ataques de aquisição de dados e controle de supervisão: onde atacante podem capturar informações e assumir o controle da rede;
- Ataques em medidores: são tipos de ataques que podem bloquear ou retransmitir comandos de controle;
- Ataques na Camada Física: ataques físicos na rede que possam gerar danos materiais;
- Injeção de dados e ataques *replay*: grandes quantidades de pacotes que visam à negação de serviço;
- Ataques baseados em rede: explorar vulnerabilidades conhecidas que a rede possui, como na pilha de protocolo TCP/IP.

Alguns ataques típicos podem afetar, negativamente, as operações de um sistema *Smart Grid*, como: DoS (*Denial of Service*) e de DDoS (*Distributed Denial of Service*), em que o objetivo é diminuir a disponibilidade do sistema, impedindo a entrega de mensagens entre dispositivos da rede. Ataques maliciosos baseados em *software* podem comprometer, direta ou indiretamente, a disponibilidade, integridade e confidencialidade do *Smart Grid*. Outro tipo de ataque é o de falsificação de identidade que permite que atacantes passem por usuários autorizados.

Ataques de pequenos furtos de senha são cometidos contra a confidencialidade dos dados. Os métodos mais comuns utilizados para esse tipo de ataque incluem: a adivinhação de senha, a engenharia social e ataques de dicionário. Ataques de espionagem, podem afetar a confidencialidade dos dados no canal de comunicação do sistema *Smart Grid*, por meio de *sniffing* de pacotes IP na LAN (*Local Area Network*). Os medidores inteligentes e dispositivos internos de uma *Smart Grid* são vulneráveis a esses ataques, podendo resultar em violação da privacidade do cliente, informações de uso, senhas e acesso administrativo ao sistema *Smart Grid,* alertam Apurva e Himanshu (2012).

3.2 REDES DEFINIDAS POR SOFTWARE E INFRAESTRUTURA

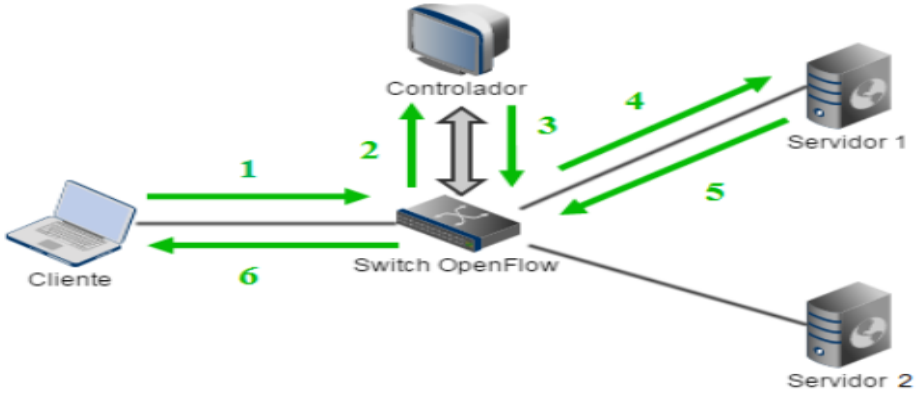
Os protocolos de rede de controle e transporte distribuídos executados internamente nos roteadores e *switches* são as principais tecnologias que permitem a informação, na forma de pacotes digitais, trafegar em qualquer lugar. Apesar da sua ampla adoção, as redes IP tradicionais são complexas e difíceis de gerenciar.

Para expressar as políticas de rede de alto nível desejadas, os operadores de rede precisam configurar cada dispositivo de rede individualmente. A reconfiguração automática e os mecanismos de resposta são praticamente inexistentes nas redes IP atuais, pois se trata de um ambiente dinâmico. Por isso, a reconfiguração é altamente desafiadora, além das redes atuais também serem integradas verticalmente.

O plano de controle (que decide como lidar com o tráfego de rede) e o plano de dados (que encaminha o tráfego de acordo com as decisões tomadas pelo plano de controle) são agrupados internamente nos dispositivos de rede, reduzindo a flexibilidade e dificultando a inovação e a evolução da infraestrutura de rede.

A rede definida por *software* ou SDN é um modelo de rede emergente promissora com capacidade de alterar as limitações das infraestruturas de rede atuais. Primeiramente, separa a lógica de controle da rede (o plano de controle) dos roteadores e *switches* que encaminham o tráfego (o plano de dados). Em segundo lugar, com a separação dos planos de controle e de dados, os *switches* de rede tornam-se dispositivos de encaminhamento simples e a lógica de controle é implementada em um controlador logicamente centralizado, simplificando a implementação de políticas, a reconfiguração e a evolução das redes. Uma visão simplificada do funcionamento dessa tecnologia apresenta-se na Figura 5.

Figura 5 - Arquitetura SDN



Fonte: Rodrigues et al. (2015)

Na Figura 5 observa-se o comportamento em uma SDN, em que (1) um cliente envia uma requisição para o servidor, (2) o *switch OpenFlow* envia a requisição para o controlador por ser a primeira requisição desse tipo e ainda não haver entrada especificada na tabela de fluxo, (3) o controlador define o que fazer com essa requisição e a devolve para o *switch OpenFlow*, (4) a mensagem é encaminhada para o servidor, (5) o servidor responde para o *switch* e (6) o *switch* responde para o cliente.

Para Yan et al. (2016), a SDN está atraindo atenção significativa da área acadêmica e empresarial. A ONF (*Open Networking Foundation*) é um consórcio sem fins lucrativos dedicado ao desenvolvimento, padronização e comercialização de SDN. Ela apresenta uma arquitetura de alto nível para SDN que é dividida, verticalmente, em três camadas funcionais principais: camada de infraestrutura, camada de controle e camada de aplicação.

A camada de infraestrutura, também conhecida como plano de dados, consiste, principalmente, em elementos de encaminhamento, incluindo *switches* físicos e *switches* virtuais (*Open vSwitch*). A camada de controle, também conhecida como plano de controle, consiste em um conjunto de controladores SDN baseados em *software* que fornecem uma funcionalidade de controle consolidada por meio de APIs abertas para supervisionar o comportamento de encaminhamento da rede por meio de uma interface aberta.

Por fim, a camada de aplicação, consiste, principalmente em aplicações que consomem serviços de comunicações da rede SDN. Exemplos desses aplicativos de serviços incluem virtualização de rede, gerenciamento de mobilidade e aplicação de segurança.

A separação do plano de controle e do plano de dados pode ser realizada por meio de uma interface de programação bem definida entre os *switches* e o controlador SDN. O controlador exerce controle direto sobre o estado nos elementos do plano de dados por meio dessa API. O exemplo mais notável desse protocolo é o *OpenFlow* (KREUTZ et al., 2015).

3.2.1 *OpenFlow*

O protocolo *OpenFlow* é um elemento da arquitetura SDN, que permite que os *switches* executem o controle dos fluxos. É um protocolo aberto que nasceu na área

acadêmica, na Universidade de *Stanford* e foi proposto para padronizar a comunicação entre os *switches* e o controlador baseado em *software* em uma arquitetura SDN, permitindo aos pesquisadores executar protocolos experimentais em rede (YAN et al., 2016).

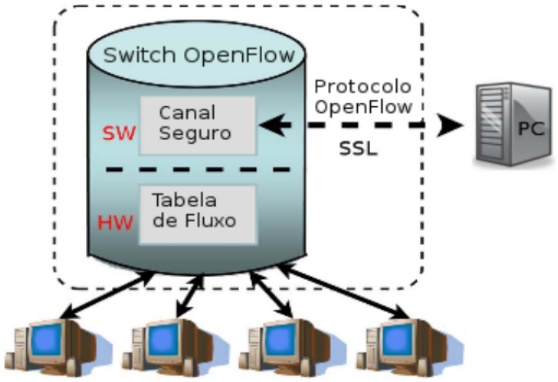
O *OpenFlow* está presente em alguns *switches* modernos, que possuem uma ou mais tabelas de regras de manipulação de pacotes (tabela de fluxo). Cada regra corresponde a um subconjunto do tráfego e executa certas ações (reencaminhamento, modificação, etc.) no tráfego. Dependendo das regras instaladas por um aplicativo do controlador, um *switch OpenFlow* pode ser instruído pelo controlador a funcionar como um roteador, um *switch*, um *firewall* ou executar outras funções, como balanceador de carga (KREUTZ et al., 2015).

De acordo com Xia et al. (2015), o *OpenFlow* é usado na maior parte das práticas de SDN que visam, principalmente, à criação de uma rede controlável por *software* e focada no controle de encaminhamento de pacotes, aproveitando o fato de que a maioria dos *switches* e roteadores *Ethernet* modernos contém tabelas de fluxo para funções essenciais de rede, como roteamento, sub-rede, proteção e análise de estatística de fluxos de dados.

Em um *switch OpenFlow*, cada entrada na tabela de fluxo possui três partes: "cabeçalho" para combinar pacotes recebidos, "ação" para definir a operação sobre os pacotes, por exemplo, aceitar ou rejeitar, e "estatísticas" do fluxo de dados. Com as informações contidas nas tabelas de fluxos, o protocolo *OpenFlow* pode oferecer serviços de manipulação na tabela de fluxo, convenientes para um controlador inserir, excluir, modificar e pesquisar as entradas da tabela de fluxo por meio de um canal TCP de forma remota.

A ideia de criar um ponto de rede separado exclusivamente para controlar a rede evidencia o conceito-chave de SDN e baseia-se na programação da rede e no controle logicamente centralizado. A Figura 6 mostra a especificação de um *switch OpenFlow,* que possui a opção de conexão segura com o controlador SDN por meio de criptografia SSL (*Secure Socket Layer)* usando o protocolo *OpenFlow*, além da tabela de fluxo que armazena estatísticas da conexão entre duas máquinas, por fim, apresenta as máquinas da rede conectadas em suas portas.

Figura 6 - Especificação do *switch OpenFlow*



Fonte: Xia et al. (2015) modificado

3.2.2 Controladores

Entre os vários projetos apresentados em Xia et al. (2015) de *softwares OpenFlow*, como o *NOX/POX, Beacon, Floodlight, Ryu, Maestro, OMNI, Trema, McNetlle* e *OpenDaylight*, o controlador NOX, provavelmente, é o controlador mais utilizado nas pesquisas relacionadas a SDN. NOX foi o primeiro controlador *OpenFlow* que permitiu que aplicativos fossem implementados com base em uma visão de rede centralizada usando nomes de alto nível em oposição a algoritmos distribuídos em endereços de baixo nível. Os aplicativos são escritos em *Python* ou C++ e são carregados dinamicamente. A infraestrutura básica e as funções críticas de velocidade do NOX são implementadas em C++.

3.2.3 NOX

NOX é um controlador para redes com uma interface programável uniforme e centralizada para toda a rede. O NOX é um controlador centralizado para adicionar e remover entradas na tabela de fluxo, ou seja, funciona como o plano de controle para *switches OpenFlow* (OH; LEE; KIM, 2011).

O NOX possui todos os recursos necessários para ser implantado em uma rede, bem como código fonte e ferramentas que permitem o desenvolvimento das próprias aplicações. O NOX pode ser incrementado em C++ e fornece uma interface abstraída para o *OpenFlow*. Também contém algumas bibliotecas incorporadas que fornecem funções de rede, como rastreamento e roteamento do *host*. O NOX depende do

OpenFlow como o protocolo de comunicação para controle de *switches* (INTERNATIONAL COMPUTER SCIENCE INSTITUTE, 2014).

3.2.4 POX

POX é um controlador de rede SDN escrito em *Python,* além da possibilidade de funcionar como um *switch OpenFlow* e ser útil para desenvolver *software* de rede em geral. Atualmente o POX se comunica com os *switches OpenFlow* 1.0 e inclui suporte especial para as extensões *Open vSwitch* (MCCAULEY, 2017).

De acordo com Kaur, Singh e Ghumman (2014), POX é um controlador *OpenFlow/SDN* de código aberto baseado em *Python* e herdado do controlador NOX. É usado para desenvolvimentos rápidos de prototipagem de novas aplicações de rede. Usando o controlador POX, é possível transformar um dispositivo *OpenFlow* em um *hub*, *switch*, balanceador de carga, dispositivos de *firewall* e outros, controlando um *switch* com tecnologia *OpenFlow*. Permite a execução de experimentos em *hardwares* reais ou em emuladores, como o *Mininet* e em infraestruturas como o GENI (*Global Environment for Network Innovations*) (GENI project, 2017).

3.2.5 GENI

O GENI disponibiliza um laboratório virtual para pesquisas e educação em redes e sistemas distribuídos, permitindo a exploração de redes em escala, promovendo inovações em ciência, segurança, serviços e aplicações em rede. O GENI permite obter recursos computacionais distribuídos espalhados nos Estados Unidos, como institutos e universidades, usando redes de camada 2 em topologias mais adequadas às experiências, permitindo a instalação de *softwares* personalizados, além do controle dos *switches* de rede no experimento, manipulando os fluxos de tráfego, e, por fim, a execução da configuração dos protocolos de camada 3 e acima pelos usuários (WHAT IS GENI, 2017).

As vantagens do GENI são várias, como a disponibilização de uma infraestrutura de experiência em larga escala. O GENI pode fornecer mais recursos do que normalmente é encontrado em qualquer laboratório, disponibilizando o acesso a centenas de recursos amplamente distribuídos, incluindo recursos de computação, como máquinas virtuais, recursos de rede como *links*, *switches* e estações base *WiMax*. Também é

possível conectividade sem utilização do protocolo IP entre os recursos, autorizando a configuração das conexões de camada 2 entre recursos de computação com a execução dos seus próprios protocolos de camada 3 e superior. Com tecnologias como *OpenFlow*, no GENI é possível programar não apenas os *hosts* finais de sua rede experimental, mas também os *switches*. Isso permite experimentar novos protocolos de camada de rede ou novos algoritmos de roteamento de IP. Ainda pode-se obter acesso exclusivo a certos recursos GENI, incluindo CPU e recursos de rede, possibilitando o controle sobre o ambiente e condições de experiências idênticas ou muito similares. Por fim, mas não menos importante, possui ferramentas de instrumentação e medição disponíveis para medidas ativas e passivas, armazenamento de dados de medição e ferramentas de visualização e análise de dados coletados (WHAT IS GENI, 2017).

Para usufruir dos recursos do GENI, é necessário se inscrever no NCSA (*National Center for Supercomputing Applications*) e solicitar uma conta gratuita do projeto GENI ou utilizá-lo por meio de uma instituição habilitada, como universidades.

Um usuário GENI pode criar projetos que organizam as pesquisas, contendo tanto as pessoas como seus experimentos. Um projeto pode ter muitos experimentadores como seus membros e um experimentador pode ser membro de muitos projetos (GENI concepts, 2017).

No GENI vários pesquisadores podem executar várias experiências ao mesmo tempo, pois o GENI utiliza o conceito de fatias. Essa fatia pode ser definida como uma unidade de isolamento para experiências. Um experimento GENI é executado em uma fatia e somente os experimentadores pertencentes a essa fatia podem realizar mudanças nos experimentos nessa fatia (GENI concepts, 2017).

Um agregado GENI fornece recursos para os pesquisadores, por exemplo, um *Rack* GENI em uma universidade é um agregado, que pode ser adicionado em uma fatia de um projeto. Os agregados podem fornecer diferentes tipos de recursos, como recursos de computação e de rede (GENI concepts, 2017).

No GENI é utilizado um documento XML de especificação de recursos, chamado de *RSpecs* GENI, para descrever recursos que os pesquisadores pretendem utilizar, como máquinas virtuais, por exemplo (GENI concepts, 2017).

Dessa forma, é possível criar e executar experimentos reservando máquinas virtuais na infraestrutura GENI, de forma gratuita e remota.

3.2.5.1 Exemplo de solicitação de recursos no GENI

Para apresentar como é feita a solicitação/reserva de recursos no GENI, será utilizado um exemplo, ilustrado em GENI (2017), com apenas dois pontos na rede, um servidor e um cliente, como mostra a Figura 7.

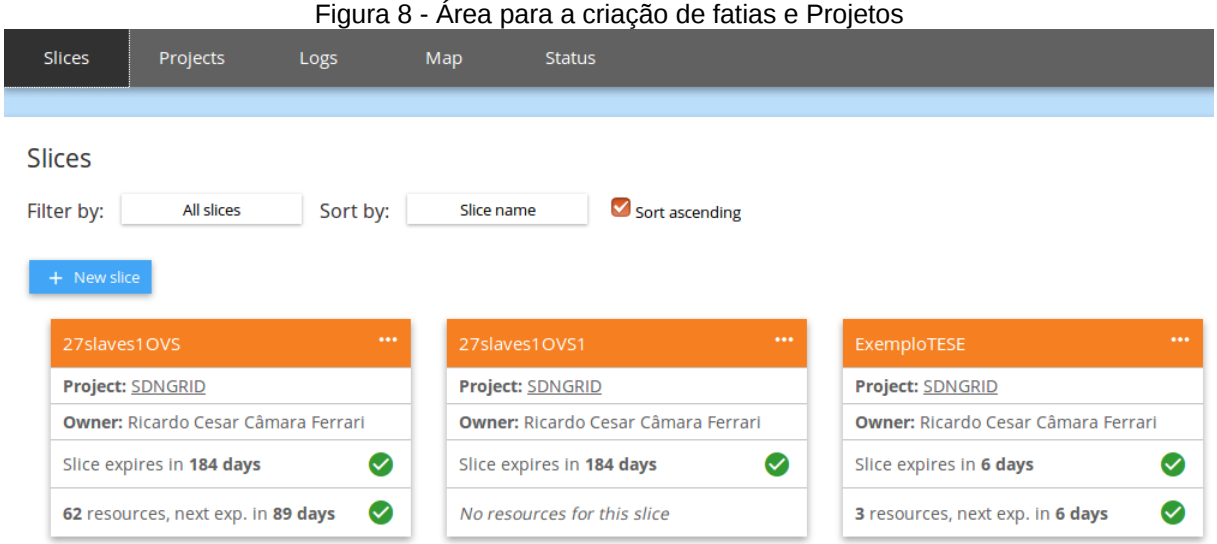
Figura 7 - Arquitetura cliente-servidor

Fonte: GENI (2017)

O primeiro passo é efetuar login no portal GENI, pressionando o botão *Use Geni* e selecionando sua instituição. Na página de *Login* de sua instituição, serão solicitados usuários e senha.

Após acessar a página do portal, será necessário criar um novo projeto e, logo depois, uma nova fatia para adicionar seus recursos, como mostra a Figura 8.

Figura 8 - Área para a criação de fatias e Projetos



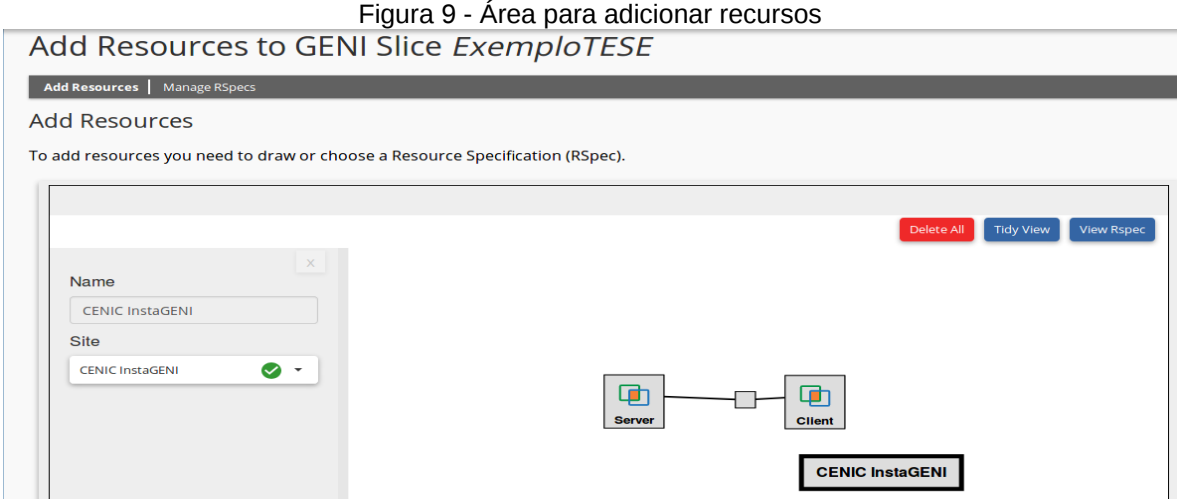
Fonte: GENI project (2017)

Esses recursos apenas poderão ser acessados depois de ser fornecida uma chave *ssh*, que fica disponível no próprio ambiente *web,* juntamente a um tutorial de apoio.

Na tela para adicionar e estruturar a rede, é possível criar uma nova estrutura

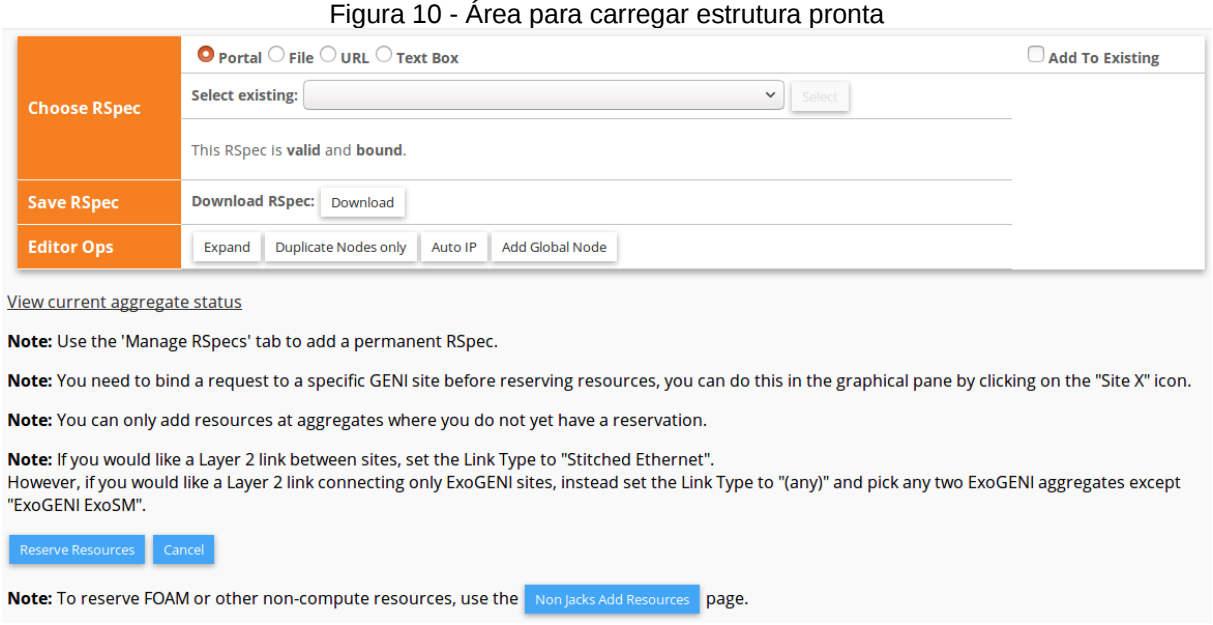
(Figura 9) ou carregar uma já existente (Figura 10). Também é possível selecionar um agregado (instituição) para que seus recursos sejam reservados, desde que haja em seu nome as vocábulos *InstaGENI* ou *ExoGENI*.

Figura 9 - Área para adicionar recursos



Fonte: GENI project (2017)

Figura 10 - Área para carregar estrutura pronta



Fonte: GENI project (2017)

Logo após da solicitação da reserva dos recursos, é preciso aguardar alguns minutos para que o processo seja finalizado, com a cor do nó verde, indicando que está disponível, como mostra a Figura 11.

Figura 11 - Recursos reservados e prontos para uso

Aggregate **CENIC InstaGENI's** Resources:

Node #1:

Status	Client ID	Component ID	Expiration	Type	Hostname
READY	Server	pc3	2017-12-26T13:07:22.000Z	emulab-xen	Server.ExemploTESE.ch-geni-net.instageni.cenic.net
Login	ssh ricardof@pc3.instageni.cenic.net -p 25331				

Interfaces		MAC	Layer 3
interface-0	pc3:lo0	0258df04e021	ipv4: 10.10.1.1

Node #2:

Status	Client ID	Component ID	Expiration	Type	Hostname
READY	Client	pc3	2017-12-26T13:07:22.000Z	emulab-xen	Client.ExemploTESE.ch-geni-net.instageni.cenic.net
Login	ssh ricardof@pc3.instageni.cenic.net -p 25330				

Interfaces		MAC	Layer 3
interface-1	pc3:lo0	02df513cb4ff	ipv4: 10.10.1.2

Link #1:

Client ID	Endpoint #0	Endpoint #1
link-0	interface-0	interface-1

Fonte: GENI project (2017)

Dessa forma, com todos os nós disponíveis (verde), inicia-se a configuração de cada máquina por meio de acesso *ssh*. A configuração depende da aplicação, pois é possível a instalação de qualquer ferramenta compatível com *Linux*. No caso da tese, foram instaladas ferramentas, como: *OpenDNP3, TCPDump* e controlador POX.

3.3 CONSIDERAÇÕES FINAIS

Para a proposta do trabalho de aplicação de SDN para auxiliar na segurança de uma *Smart Grid*, foi detalhado como as redes são empregadas e suas principais tecnologias, como os sistemas *SCADA* para controle e gerenciamento de redes *Smart Grid*, o protocolo DNP3 normalmente utilizado nesse tipo de ambiente e as vulnerabilidades que essa rede possui, mostrando que é uma área que necessita de mais estudos.

A SDN, por sua vez, possui, basicamente, o controlador, capaz de capturar dados e enviar comandos para o *switch* e o protocolo *OpenFlow* que possibilita essa comunicação entre o controlador e o *switch* para definição de regras e monitoramento na tabela de fluxo.

No final do capítulo, foi apresentado o GENI, uma infraestrutura que permite que sejam criadas redes virtuais com recursos de comunicação DNP3

e *OpenFlow*. Com isso, o capítulo seguinte apresenta a combinação entre *Smart Grid* e SDN, utilizando a infraestrutura GENI para a criação dos ambientes de testes.

4 ARQUITETURA: SDN SOBRE *SMART GRID*

Neste capítulo serão apresentados os planos de testes para a realização dos experimentos 1, 2 e 3.

Em cada um dos experimentos, foram utilizados recursos diferentes, conforme a evolução do trabalho, essas ferramentas são descritas a seguir.

4.1 *VIRTUALBOX*

O *VirtualBox* é um *software* de virtualização atualmente executado em sistemas *Windows, Linux, Macintosh* e *Solaris* (VIRTUALBOX, 2017b).

Na referência *VirtualBox* (2017ª) são descritas as principais características do *VirtualBox*, como a portabilidade, permitindo que seja possível executar máquinas virtuais criadas em um *host* A e depois em um *host* B com um sistema operacional diferente.

4.2 *WIRESHARK*

Como descrito em *Wireshark* (2017), é um analisador de protocolo de rede que permite o monitoramento do que está ocorrendo na rede em um nível microscópico.

O *Wireshark* possui alguns recursos que incluem as seguintes características (WIRESHARK, 2017):

- Inspeção de centenas de protocolos;
- Análise de captura em tempo de execução e *offline;*
- Os dados de rede capturados podem ser visualizados por meio de uma interface gráfica ou *prompt* de comando;
- Filtros de exibição com vários recursos.

4.3 *AXON TEST SIMULATOR*

Detalhada em Axon Group (2017), a ferramenta foi desenvolvida para análise e teste de protocolos de telecontrole, permitindo simular conexões com diferentes protocolos de comunicação, tanto mestre quanto escravo.

É um produto da *Axon Group* Ltda e possui vários recursos e protocolos para análise e teste de diferentes IEDs ou centros de controle de forma simples e rápida usando protocolos como, DNP3 LAN/WAN/serial (mestre/escravo), IEC 60870-5-104/60870-5-101/60870-5-103 (mestre/escravo), *Modbus* (mestre/escravo) e OPC (*Object Linking and Embedding (OLE) for process control OPC*).

4.4 *TCPREPLAY*

O *Tcpreplay* é um conjunto de ferramentas que permite usar o tráfego previamente capturado no formato *libpcap* para testar uma variedade de dispositivos de rede. Permite classificar o tráfego para o cliente ou para o servidor, reescrever os cabeçalhos da camada 2, 3 e 4 e, finalmente, reproduzir o tráfego de volta para a rede por meio de outros dispositivos (TCPREPLAY, 2017)

4.5 OVS (OPEN VSWITCH)

Open vSwitch é um *switch* de *software* multicamada licenciado sob a licença *Open Source Apache* 2. É adequado para funcionar como um *switch* virtual em ambientes com máquinas virtuais. Além de expor interfaces de controle e visibilidade padrão para a camada de rede virtual, ele foi projetado para suportar a distribuição em vários servidores físicos. *Open vSwitch* oferece suporte a várias tecnologias de virtualização baseadas em *Linux*, incluindo *Xen/XenServer*, KVM (*Kernel-based Virtual Machine*) e *VirtualBox* (OPEN vSwitch, 2017).

4.6 *TCPDUMP*

A ferramenta *Tcpdump* é um analisador de pacotes que apresenta os cabeçalhos dos pacotes e, conforme apresentado por Jacobson, Leres e Mccanne (2017), imprime uma descrição do conteúdo dos pacotes de uma interface de rede. Também pode ser executado para que seja possível salvar/ler os pacotes em um arquivo para análise posterior. Além de poder ser executado até que seja interrompido por um sinal de interrupção ou por um número especificado de pacotes.

Quando o *Tcpdump* termina a captura de pacotes, há o registro das contagens de pacotes capturados, pacotes recebidos pelo filtro e pacotes perdidos. Com essa

ferramenta é possível analisar e monitorar o tráfego de uma rede para entender certos comportamentos e resolver determinados problemas.

4.7 *OPENDNP3*

O *OpenDNP3* é uma biblioteca portátil, escalável e com sua implementação testada com protocolo DNP3. A biblioteca é otimizada para execução de simulações de dispositivos escravos e mestres (DNP3, 2013b).

O *DNP Test Set* (TS) é uma ferramenta para verificar comunicações DNP entre escravos e mestres. A instalação DNP TS inclui configurações padrão para configurações mestre e escrava, porém, configurações personalizadas são realizadas por meio de edição ou replicação dos arquivos de configuração padrão (DNP3, 2013a).

4.8 EXPERIMENTOS

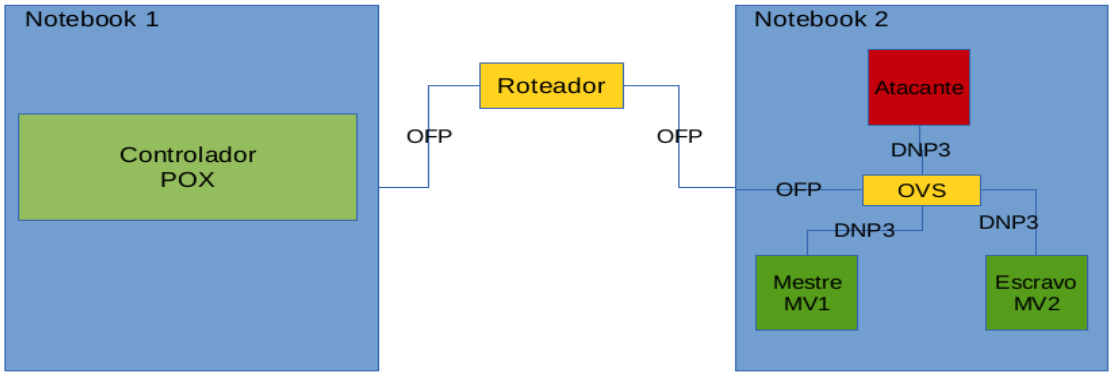
Após um conhecimento prévio das ferramentas necessárias para o desenvolvimento do trabalho, são realizados três experimentos:

- Experimento 1: ataques em mestre e escravos da *Smart Grid;*
- Experimento 2: fluxo entre OVS e controlador;
- Experimento 3: arquitetura e solução para detecção de anomalias e contramedidas.

4.8.1 Experimento 1: ataques em mestre e escravo da *Smart Grid*

No primeiro experimento, foram utilizados dois notebooks, sendo um o controlador da rede SDN (notebook 1) e outro com três máquinas virtuais gerenciadas pela ferramenta *Virtualbox*, MV1 (Máquina Virtual 1), MV2 (Máquina Virtual 2) e MV3 (Máquina Virtual 3) (notebook 2), representando as máquinas mestre e escravo de uma *Smart Grid* e um atacante, respectivamente. O objetivo principal do experimento consiste em verificar o fluxo DNP3 entre as duas máquinas virtuais, MV1 e MV2, quando estão sendo atacadas pela máquina Atacante. A Figura 12 apresenta o ambiente utilizado para os experimentos.

Figura 12 - Ambiente de testes do experimento 1



Fonte: elaboração do próprio autor

4.8.1.1 Equipamentos

Os equipamentos utilizados para a criação do ambiente de testes foram dois notebooks, com as seguintes configurações:

- *Notebook* 1:
  - Sistema operacional *Lubuntu Desktop* 13.04;
  - *Kernel Linux 3.8.0-19-generic*;
  - 3GB de memória RAM;
  - Processador *Intel Centrino 2 Core 2 Duo P8400 2.26 GHz*.
- *Notebook* 2:
  - Sistema operacional *Ubuntu Desktop* 10.04;
  - *Kernel Linux 2.6.32-51-generic- pae*;
  - 4GB de memória RAM;
  - Processador *Pentium Dual Core* T4300 2.10 GHz.

Dessa forma, as máquinas virtuais, foram criadas com as seguintes configurações:

- *Master / Slave*:
  - Sistema operacional *Windows 7 Professional*;
  - 512 MB de memória RAM.
- Atacante:
  - Sistema operacional *Kali*;
  - 512 MB de memória RAM.

As três máquinas virtuais estão ligadas na rede sem fio pelo modo *bridge* do *VirtualBox*. O modelo do roteador *wireless* é TP-WR541G.

4.8.1.2 Ferramentas

Para os testes realizados, foram utilizados os *softwares*:

- *VirtualBox;*
- *Wireshark;*
- *Axon Test Simulator;*
- *DNP3 Test Set;*
- *Tcpreplay.*

4.8.1.3 Plano de testes

Na realização dos testes, foram utilizados os notebooks e as três máquinas virtuais, descritos na subseção 4.8.1.1 (Equipamentos), sendo as três máquinas virtuais hospedadas no notebook 2, o *Master, o Slave* e o Atacante. Com isso, as máquinas *Master e Slave* produziram fluxo DNP3, com a máquina *Master* utilizando a ferramenta *Axon Test Simulator* e a máquina *Slave* a ferramenta *OpenDNP3 Test Set.* Dessa forma, a máquina Atacante utilizou o *Wireshark* para capturar o tráfego DNP3 e salvar como arquivo .pcap para ser replicado com a ferramenta *Tcpreplay*.

A ferramenta *Tcpreplay* foi utilizada com os seguintes parâmetros:

- *tcpreplay –multiplier=X –loop=0 –intf1=eth0 arquivo.pcap*

A letra X no comando é a velocidade de repetição do fluxo original, ou seja, X vezes a velocidade do fluxo original, sendo que foram utilizados os valores 5, 10, 50, 100, 500, 1000, 2000, 5000 e 15000.

O *arquivo.pcap* é o fluxo armazenado com a ferramenta *Wireshark*, em que foram utilizados dois arquivos .pcap, *final_slave.pcap* para os ataques na máquina *Slave* e *finaldnp3.pcap* para ataques na máquina *Master*.

Os parâmetros *loop* e *intf1* definem que o ataque não terminará e a interface de saída é *eth0*, respectivamente.

O fluxo original capturado pelo *Wireshark* e repetido pelo *Tcpreplay* possui 9440 pacotes (cerca de 7,38 *Mbytes*) enviados em 253,61 segundos.

4.8.2 Experimento 2: fluxo entre OVS e controlador

No segundo experimento, realizaram-se testes para verificar o fluxo entre o OVS e o controlador.

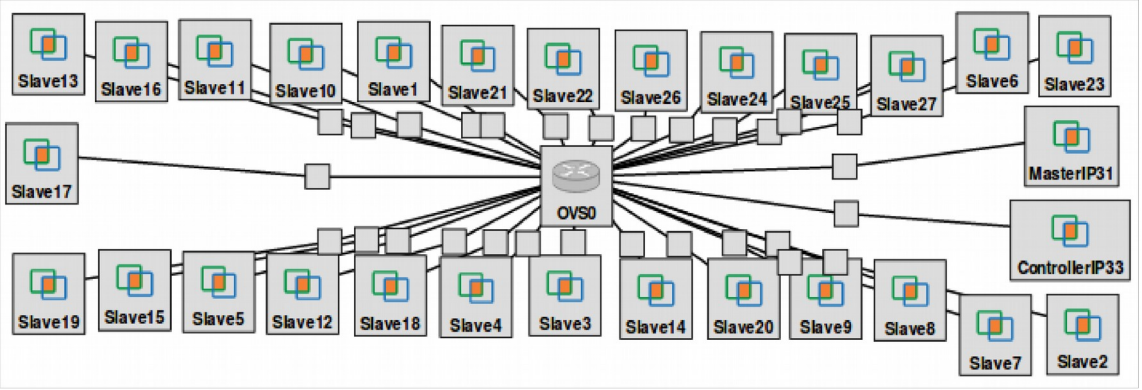
4.8.2.1 Equipamentos

Como infraestrutura para os testes, foi utilizado o GENI, com uma rede com composta por 30 nós virtuais hospedados em servidores físicos na universidade de Kentucky (EUA). A configuração dos nós virtuais segue a seguinte distribuição:

- 1 controlador POX;
- 1 OVS;
- 1 mestre (*OpenDNP3*);
- 27 escravos (*OpenDNP3*).

Cada máquina virtual com o ambiente Ubuntu 16.04.1 LTS (*Long Term Support*) (*GNU/Linux 3.13.0-33-generic x86-64*), processador *Intel Xeon X5650 2.67GHz* e 1 GB de memória RAM. A Figura 13 apresenta a configuração do ambiente dos testes.

Figura 13 - Ambiente de testes do experimento 2



Fonte: elaboração do próprio autor

4.8.2.2 Ferramentas

Além do GENI, foram utilizados os *softwares, OVS 2.3.1, Tcpdump 4.9.0, OpenDNP3 1.1.0-RC1* e o controlador *POX 0.1.0-betta.* A versão *Betta* foi escolhida devido ao recurso de estatística que ela possui, mas nada impede de ser codificada em uma versão estável.

4.8.2.3 Plano de testes

Para realizar os testes, as 29 máquinas foram interligadas por meio do OVS. Com isso, o mestre gera fluxo DNP3 contínuo para cada conexão individual de aproximadamente 62 pacotes/min e 4830 *bytes*/min e as máquinas escravas respondem as solicitações com 31 pacotes/min e 2738 *bytes*/min, com a ferramenta *OpenDNP3*. Por fim, o controlador da rede SDN está preparado para receber os fluxos necessários para as comunicações entre as máquinas escravas e o mestre da *Smart Grid*. No controlador utilizou-se o *Tcpdump* para capturar o tráfego entre ele e o OVS, para que fosse analisado posteriormente. Foram realizados dois testes, com 9 e 27 escravos, produzindo 2 capturas de fluxos no *Tcpdump*. De acordo com Lu et al. (2011), os requisitos do *delay* para entrega de mensagem pode variar de acordo com a aplicação (Tabela 2).

Tabela 2 - Requisitos de *delay* para entrega de mensagens.

Proteção	3 – 16 ms
Monitoramento em tempo real	16 – 100 ms
Baixa velocidade	>= 100 ms

Fonte: Lu et al. (2011).

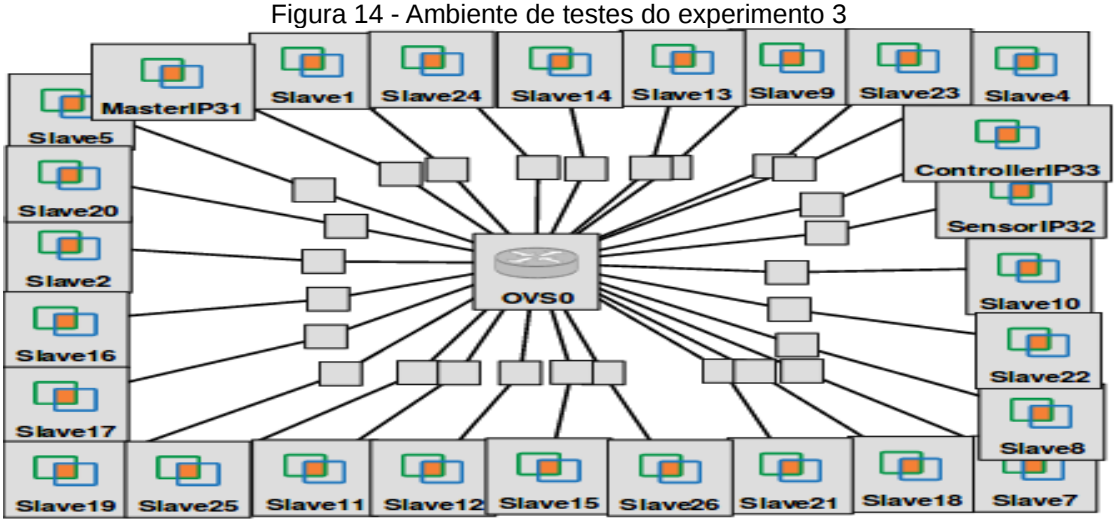
4.8.3 Experimento 3: arquitetura e solução para detecção de anomalia e contramedidas

Por fim, no último experimento, o objetivo é apresentar uma arquitetura em camadas para uma aplicação SDN em *Smart Grid* e analisar o funcionamento do componente *flow_stats* modificado para a detecção e contramedidas de ataques DDoS, tanto para ataques originados de máquinas invasoras, como de escravos da rede *Smart Grid*, possibilitando identificar a eficiência da aplicação como um sistema de detecção de intrusos.

4.8.3.1 Equipamentos

A Figura 14 apresenta o ambiente utilizado para testes. Para a realização, foi utilizado novamente o ambiente GENI, sendo 24 máquinas escravas, 1 OVS, 1 sensor, 1 mestre e 1 controlador POX.

Figura 14 - Ambiente de testes do experimento 3 Fonte:



elaboração do próprio autor

4.8.3.2 Ferramentas

Além do GENI, foram utilizados os *softwares*, OVS 2.3.1, *OpenDNP3 1.1.0-RC1* e o controlador *POX 0.1.0-betta*.

4.8.3.3 Plano de testes

Para realizar o experimento, foram utilizadas 27 máquinas interligadas por meio do OVS. Com isso, o mestre gera fluxo DNP3 contínuo para 16 escravos de aproximadamente 62 pacotes/min e 4830 *bytes*/min e as máquinas escravas respondem as solicitações com 31 pacotes/min e 2738 *bytes*/min, com a ferramenta *OpenDNP3,* possibilitando com isso o cálculo dos desvios padrões (Dn) em 12 intervalos de 2 horas (24 horas), como pode ser visto na Figura 15.

Figura 15 - Cálculo e análise de fluxos do *switch* e das máquinas

Fonte: elaboração do próprio autor

O controlador da rede SDN está preparado para monitorar os fluxos necessários para as comunicações entre as máquinas escravas e o mestre da *Smart Grid* para os cálculos dos desvios padrões. Uma máquina sensor vasculha a rede durante o período de monitoramento, a fim de auxiliar a detecção de novas máquinas na rede. No controlador foi empregado o POX para monitorar o tráfego entre as máquinas, para que fosse identificada máquinas intrusas na rede, possíveis ataques DDoS e anomalias nos fluxos individuais com o dobro do desvio padrão.

Após as finalizações dos cálculos dos desvios padrões D1-D12, no período de 24 horas, com início às 10 horas, o controlador é capaz de iniciar o monitoramento do fluxo no *switch*, identificando possíveis máquinas invasoras, ataques e anomalias nos fluxos.

Para os cálculos dos desvios padrões, foi definido um tempo de leitura da tabela de fluxo do OVS de 10 seg., com 720 leituras por intervalo, para determinar os conjuntos de 720 valores de pacotes e *bytes* (x_i) utilizados para os cálculos dos desvios padrões. Dessa forma, para os cálculos dos desvios padrões o primeiro passo é encontrar a média aritmética (que designaremos por média) de pacotes e *bytes,* utilizando a equação (1), mais detalhes podem ser encontrados em BUSSAB e MORETTIN (2017):

$$\overline{x} = \frac{x_1 + x_2 + \cdots + x_n}{n} = \frac{\sum_{i=1}^{n} x_i}{n}, \qquad (1)$$

Depois de encontrar as médias de pacote e *bytes* para cada intervalo, o mesmo acontece com o cálculo da variância representada pela equação (2):

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1} \quad (2)$$

Por fim, é calculado o desvio padrão aplicando a raiz quadrada na variância, como mostra a equação (3):

$$s = \sqrt{\text{variância}} = \sqrt{s^2} \quad (3)$$

Como mencionado, todo esse processo é realizado para cada intervalo (D1, D2, … e D12) e em cada fluxo de comunicação individual, além do cálculo de todo o fluxo do *switch*.

Os valores definidos, como tempo de leitura, quantidade de leituras, IP do mestre, número de desvios padrões, tempo de bloqueio, IP do sensor e número de desvios padrões para os fluxos e *switch* podem ser alterados em um arquivo de configuração.

Foram realizados dois ataques DDoS no mestre da rede *Smart Grid*, sendo o primeiro partindo de 8 máquinas intrusas e o segundo a partir de 8 escravos.

4.9 CONSIDERAÇÕES FINAIS

Após a apresentação dos capítulos dos trabalhos relacionados e das descrições das tecnologias *Smart Grid* e SDN, foi apresentada a proposta para a combinação entre as duas tecnologias, juntamente ao detalhamento dos ambientes de testes e as ferramentas utilizadas.

Três ambientes de testes foram propostos neste capítulo, o primeiro com ataques *Tcpreplay* no mestre e no escravo da *Smart Grid*, com o intuito de analisar o comportamento de cada um durante o ataque. O segundo ambiente de testes para verificar o fluxo de mensagens *OpenFlow* entre o controlador e o *switch,* e o terceiro que apresenta uma solução de segurança para *Smart Grid* com o uso de SDN, em que foi modificado um componente do controlador POX para controle e gerenciamento de fluxos DNP3.

Contudo, após a definição dos ambientes de testes, no próximo capítulo são apresentados os resultados de cada testes e, posteriormente realizada um conclusão final sobre os objetivos de segurança alcançados.

5 RESULTADOS

Após os testes com os parâmetros apresentados, foram obtidos os resultados dos três experimentos:

- Experimento 1: resultados para os ataques em mestre e escravo da S*mart Grid;*
- Experimento 2: resultados do fluxo entre OVS e controlador;
- Experimento 3: resultados da solução para detecção de anomalia e contramedidas.

5.1 EXPERIMENTO 1: RESULTADOS PARA OS ATAQUES EM MESTRE E ESCRAVO DA *SMART GRID*

A Tabela 3 apresenta as durações dos ataques que o mestre e o escravo sofrem para cada velocidade. O tempo medido foi em segundos e de forma aleatória.

Tabela 3 - Duração para cada tipo de ataque

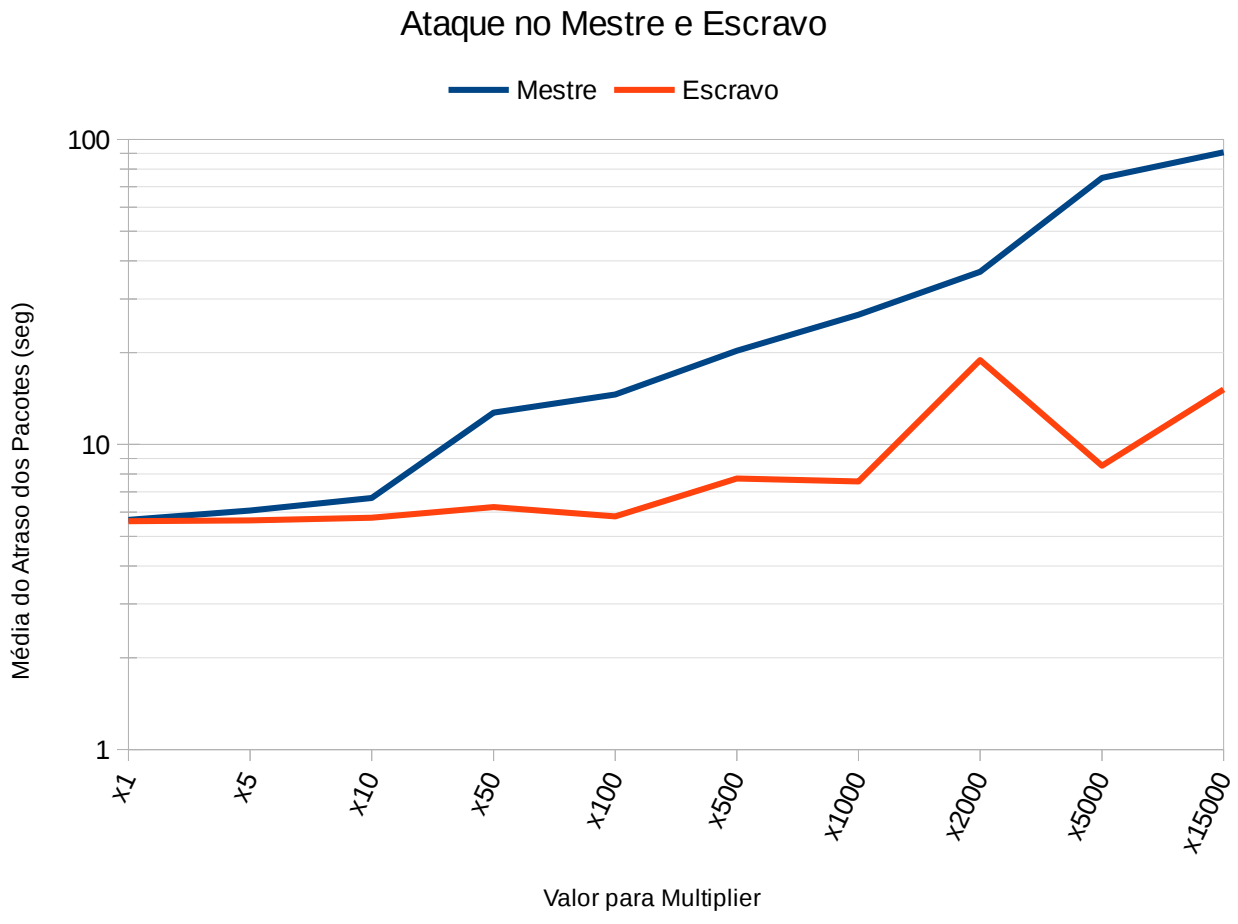
Durações dos Ataques (seg.)		
Velocidade do Fluxo	**Master**	**Slave**
x1	339	588
x5	370	603
x10	488	664
x50	953	741
x100	613	599
x500	466	681
x1000	558	801
x2000	700	1457
x5000	598	460
x15000	816	878

Fonte: elaboração do próprio autor

Dessa forma, foram realizados ataques para investigar o comportamento do mestre e do escravo quando eram atacados, no contexto de troca de pacotes DNP3. Os resultados são apresentados na Figura 16.

A Figura 16 apresenta um aumento no tempo de troca de pacotes entre o mestre e o escravo conforme aumenta a velocidade do ataque no mestre. O mesmo não ocorre quando o ataque é no escravo, não tendo um impacto tão visível como o ataque direcionado para o mestre. Porém, uma variação nos ataques direcionados para o escravo chama atenção, quando o valor do *Multiplier* é igual a 2000, causando um aumento da média do atraso na troca de pacotes entre o mestre e o escravo.

Figura 16 - Média do atraso na troca de pacotes entre o Mestre e o Escravo com ataques diferentes

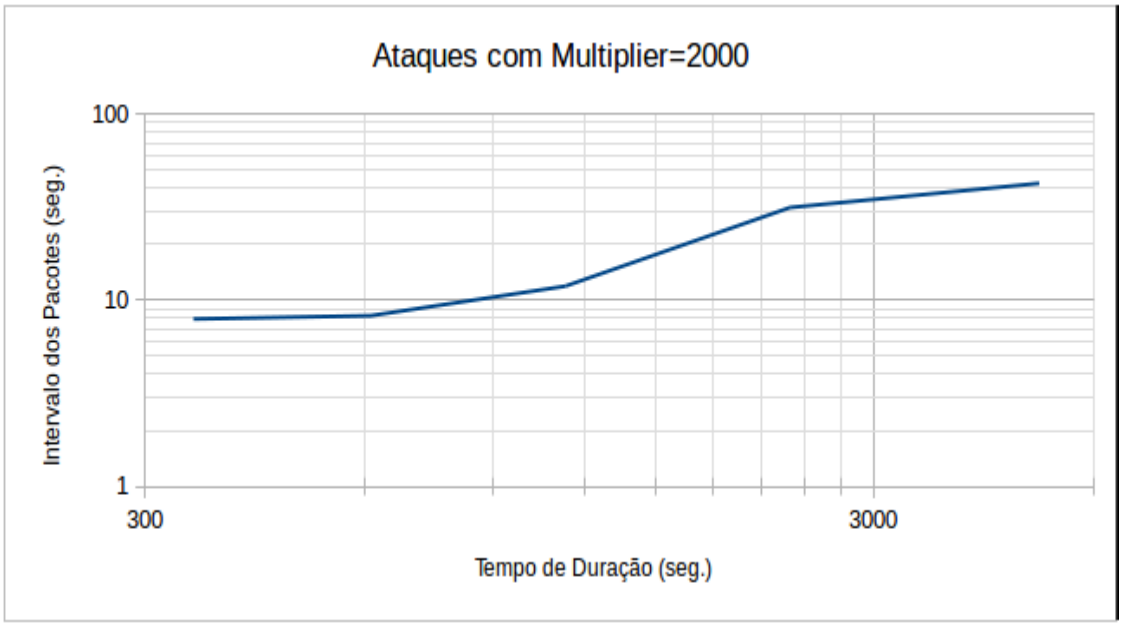


Fonte: elaboração do próprio autor

Analisando a Tabela 3, observa-se que a duração desse ataque (1457,3971 segundos) foi maior em relação aos outros, isso motivou a realização de ataques no

escravo de forma diferente, fixando a velocidade em 2000 e variando o tempo de duração do ataque. A Figura 17 apresenta os resultados dos testes.

Figura 17 - Média do atraso na troca de pacotes entre o mestre e o escravo com o mesmo ataque e tempos diferentes



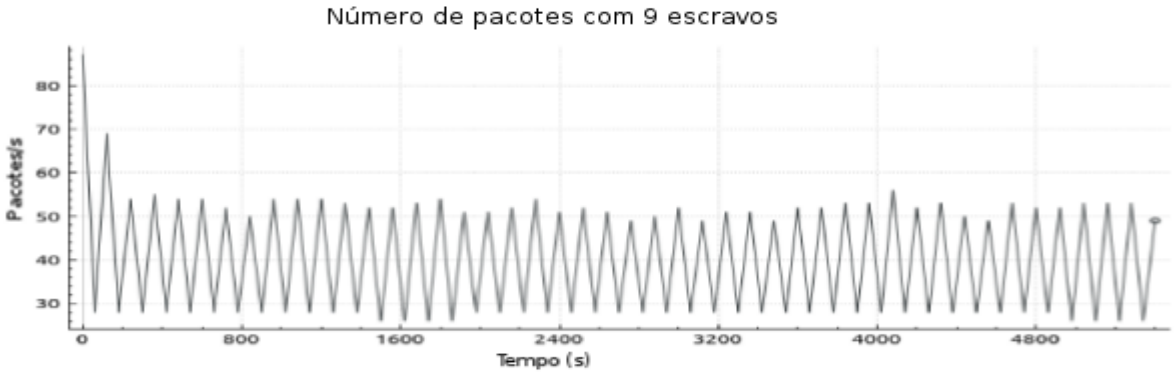
Fonte: elaboração do próprio autor

A Figura 17 apresenta uma variação significativa dependendo da duração do ataque, conforme a duração do ataque aumenta, a média do atraso na troca de pacotes entre o mestre e o escravo também aumenta. O ataque com duração de 10960,65 segundos foi suficiente para derrubar a máquina virtual que simulava o escravo, revelando que, em um ataque distribuído o tempo para derrubar um nó na rede pode ser menor e, comprometer o monitoramento da rede.

5.2 EXPERIMENTO 2: RESULTADOS DO FLUXO ENTRE OVS E CONTROLADOR

Após a realização dos testes, foram obtidos resultados mostrando as características de comunicação entre o controlador e o *switch* em cada um dos experimentos, com o número de pacotes e tempo de captura, permitindo avaliar se o fluxo entre o controlador e o *switch* é afetado com um número maior de escravos na rede *Smart Grid*. As Figuras 18 e 19 mostram as capturas do número de pacotes de transmissões.

Figura 18 - Captura com 9 escravos.



Fonte: Elaboração do próprio autor.

Figura 19 - Captura com 27 escravos.

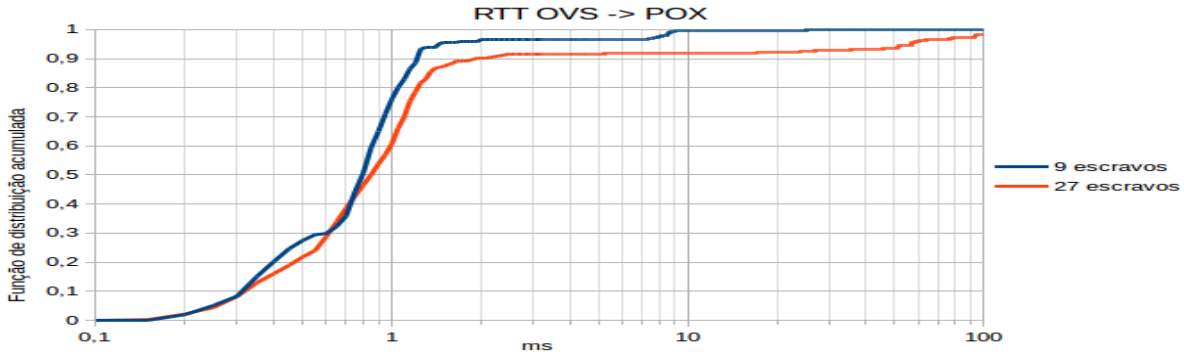
Fonte: Elaboração do próprio autor.

As Figuras 18 e 19 apresentam picos de pacotes/s no fluxo entre o *switch* e o controlador no início da comunicação entre os escravos e o mestre. Isso ocorre devido à quantidade de mensagens *OpenFlow* que é necessária para a escrita das novas rotas na tabela de fluxo do *switch*. Para isso, o *switch* envia mensagens *OFPT_PACKET_IN* para o controlador, sendo essa primitiva utilizada quando não é possível encontrar na tabela de fluxo do *switch* uma entrada correspondente a ele (SILVA, 2013), como é o caso das novas comunicações entre o mestre e os escravos. Outro tipo de mensagem enviada é o *OFPT_PACKET_OUT*, que funciona de maneira complementar, enviando um pacote do controlador para o *switch*, com o objetivo de atualizar a tabela de fluxo sempre que for identificada uma nova entrada de dados. Além dos picos identificados nas Figuras 18 e 19, observa-se que, mesmo com o aumento de escravos na rede *Smart Grid,* a quantidade de pacotes *OpenFlow*

continua abaixo dos 60 pacotes/s, com aumento apenas no início da comunicação, em consequência do maior número de regras na tabela de fluxo do *switch*.

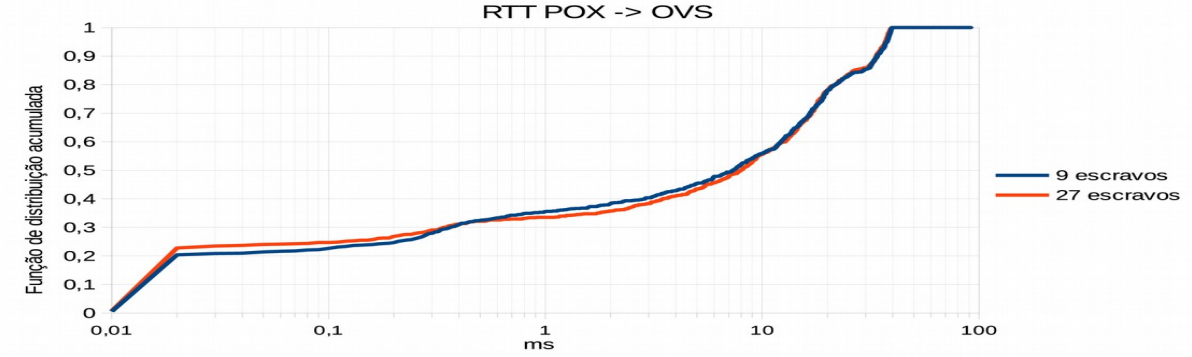
Outro tipo de análise realizada, apresentada nas Figuras 20 e 21, foi a medição do RTT (tempo para enviar um pacote de requisição de eco mais o tempo para tê-lo de volta) entre o *switch* e o controlador, com o intuito de avaliar o comportamento da rede com diferentes números de escravos. A análise identificou que os RTTs do OVS para o POX tiveram um comportamento similar, tanto para 9 escravos, como para 27 escravos. Na Figura 20, observa-se que, para 90% dos casos, o RTT com 9 escravos (curva azul) é menor ou igual do que 1,2 ms, enquanto para 27 escravos (curva vermelha) é menor ou igual a 2 ms. Na Figura 21, as curvas são muito similares, indicando que o RTT do controlador (POX) para o *switch* (OVS) não é influenciado pela quantidade de escravos. Além disso, a duração máxima do RTT é menor do que 40 ms, mostrando que o atraso na comunicação ficou dentro do intervalo para aplicações de segurança, que é de 3 – 16 ms.

Figura 20 - Função Distribuição Acumulada do RTT do OVS para o POX



Fonte: elaboração do próprio autor

Figura 21 - Função Distribuição Acumulada do RTT do POX para o OVS

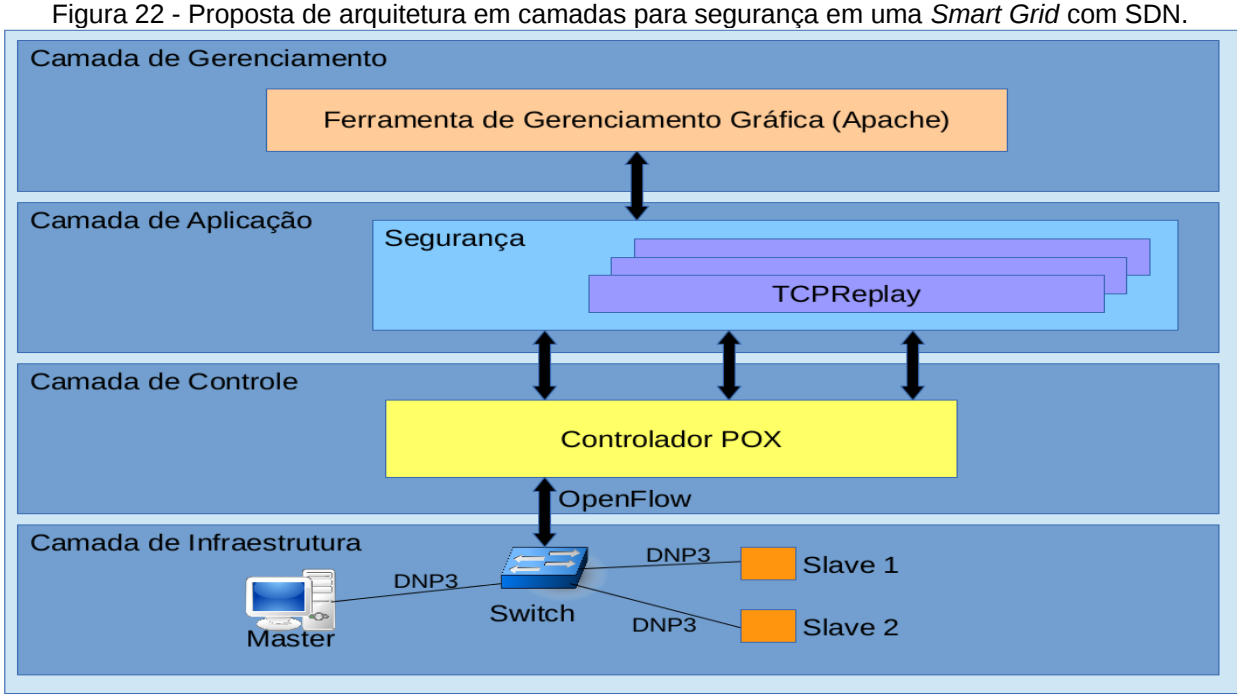


Fonte: elaboração do próprio autor

5.3 EXPERIMENTO 3: RESULTADOS DA SOLUÇÃO PARA DETECÇÃO DE ANOMALIA E CONTRA MEDIDAS

Os resultados anteriores motivaram a proposta de uma arquitetura em camadas que permitisse a implementação de uma solução de segurança para uma *Smart Grid* com o auxilio de uma SDN, resultando na arquitetura da Figura 22.

Figura 22 - Proposta de arquitetura em camadas para segurança em uma *Smart Grid* com SDN.



Fonte: Elaboração do próprio autor.

A arquitetura proposta é dividida em quatro camadas: Infraestrutura, Controle, Aplicação e Gerenciamento.

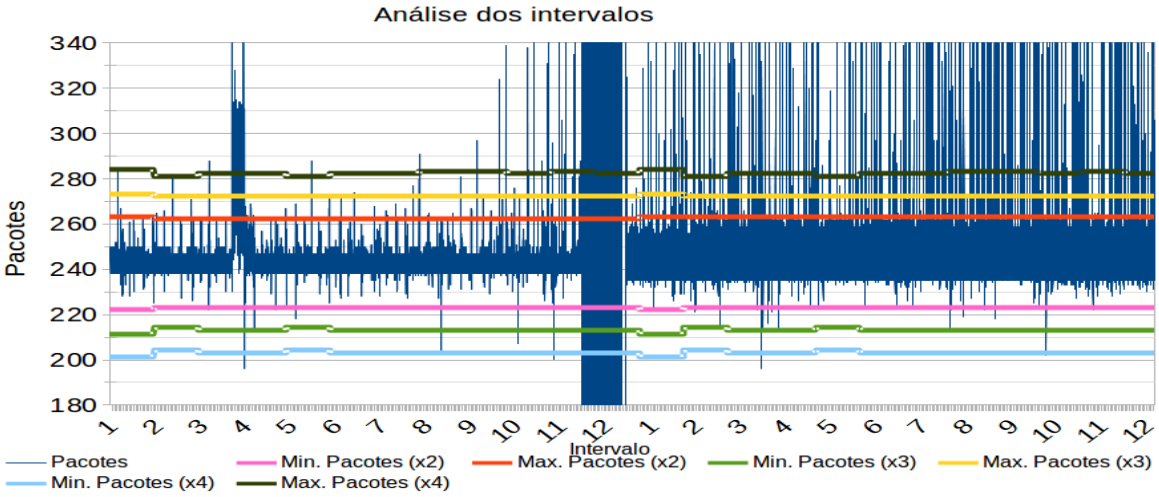
A camada de Infraestrutura representa a parte física da rede *Smart Grid* que possui um *switch* com suporte *OpenFlow* e está ligado à camada de Controle, que, por sua vez, possui um controlador de rede SDN. Dessa forma, o controlador monitora o fluxo de pacotes que trafega na *Smart Grid*. A camada de Aplicação representa os recursos oferecidos pela arquitetura, como um mecanismo de segurança para ataques *TCPReplay*. Por fim, a camada mais alta de Gerenciamento contém a aplicação gráfica para operar as camadas mais baixas, como configuração do controlador e *Smart Grid*.

Após os testes, foram obtidos os resultados das análises realizadas pelo controlador nos intervalos de 1 a 12 por duas vezes, monitorados por 48 horas com início às 10:00 horas (Figura 15), após os cálculos dos desvios padrões. O fluxo total no OVS

entre o mestre e os 16 escravos resultaram no mínimo de 28 e máximos de 489 falsos positivos nos períodos 5 e 12 respectivamente, usando os respectivos valores de mínimo e máximo de pacotes, definidos a partir dos desvios padrões calculados pelo dobro, tanto para mais como para menos.

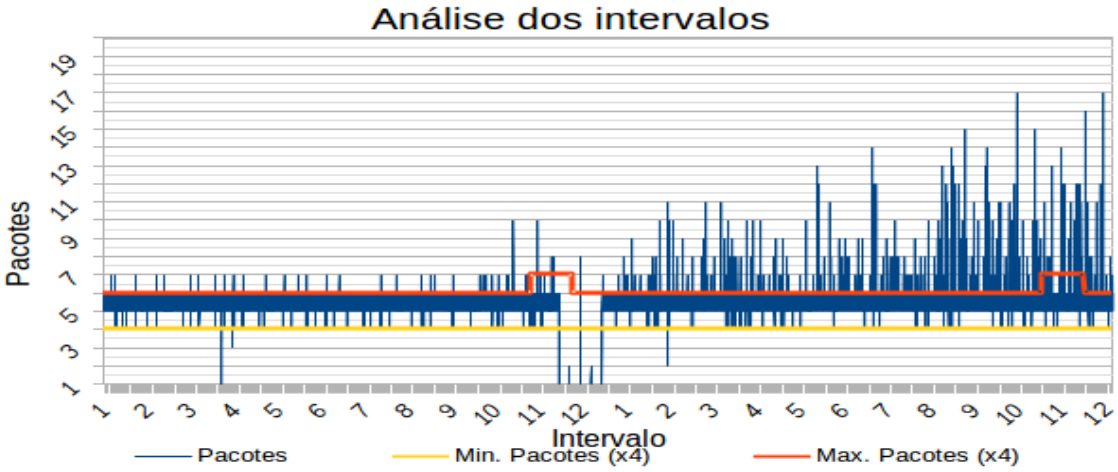
Observou-se, que no intervalo 12, o número de anomalias foi alto, graças ao segundo ataque DDoS realizado por escravos, provocando seguidas execuções do componente *host_tracker*.

Figura 23 - Monitoramento do total de pacotes no OVS



Fonte: elaboração do próprio autor

Figura 24 - Monitoramento de pacotes máquina 15



Fonte: elaboração do próprio autor

Contudo, foram gerados os gráficos das 48 horas de monitoramento, apresentados nas Figuras 23 e 24, apenas para os pacotes, pois o comportamento do fluxo de *bytes* é muito semelhante.

Assim, pode-se verificar o comportamento do fluxo em todo período de monitoramento e o momento que ocorreram as anomalias, tanto no OVS como na máquina 15 (escravo atacante). As Figuras apresentam o número mínimo e máximo de pacotes, com o dobro, triplo e quádruplo do desvio padrão na Figura 23 e apenas o quádruplo na Figura 24 para uma melhor visualização.

A Figura 23 apresenta o comportamento do fluxo no decorrer do tempo, mostrando que próximo as 15 horas (intervalo 3) e 9 horas (intervalo 11 e 12) das primeiras 24 horas ocorreram duas anomalias no fluxo por consequência de dois ataques DDoS com duração de, aproximadamente, 35 minutos e 112 minutos respectivamente.

Nas últimas 24 horas, o fluxo apresentou um comportamento anômalo nas Figuras 23 e 24, isso se deve ao aumento do processamento do componente *host_tracker* que foi modificado para auxiliar na detecção de máquinas intrusas e ocasionou atrasos na leitura da tabela de fluxo, apresentando valores mais altos para o número de pacotes, contribuindo para um número maior de falsos positivos. Esse comportamento foi identificado em todos os fluxos.

Também pode-se observar que a Figura 23 apresenta anomalias com valores altos no primeiro ataque e altos e baixos no segundo ataque. Isso ocorreu em virtude do primeiro ataque ser realizado por máquinas invasoras que apenas acrescentaram pacotes na rede durante o ataque. Já em relação ao segundo ataque, os atacantes eram escravos que já estavam se comunicando, dessa forma, o mestre continuou tentando conectar-se com os escravos bloqueados, o que acarretou um fluxo aleatório no *switch* devido aos bloqueios das portas dos escravos. Ainda na Figura 23, os picos no início e no final do primeiro ataque são resultados da continuação do ataque DDoS pelas máquinas invasoras. O tempo de bloqueio das portas foi definido em 30 minutos.

Na Figura 24, são mais evidentes os bloqueios das portas, justamente pela interrupção do fluxo decorrente do bloqueio.

No primeiro ataque, a ausência de fluxo para a máquina 15 é sequela da limpeza da tabela de fluxo para liberar as portas que estavam bloqueadas. Porém, no segundo ataque, a ausência de fluxo é, exclusivamente, pelo bloqueio de sua porta pelo tempo total de 2 horas, os 3 picos no período de bloqueio é pelo desbloqueio de sua porta após 30 minutos, bloqueando imediatamente depois de identificar a

continuidade do ataque. Para os dois ataques, o controlador evitou 99% dos pacotes enviados, com os bloqueios das portas dos atacantes.

5.4 CONSIDERAÇÕES FINAIS

A utilização de redes definidas por *software* na área de segurança e monitoramento é cada vez mais presente em ambiente críticos, como nas *Smart Grid*, devido à flexibilidade das regras em um controlador poderem ser definidas independente do hardware, desde que o *switch* tenha suporte *OpenFlow*.

A pesquisa apresentou, no experimento 1, a fragilidade de uma *Smart Grid* quando é atacada, podendo comprometer seu funcionamento, além de analisar no experimento 2 o desempenho que uma aplicação de redes definidas por *software* pode ter em uma aplicação em *Smart Grid*.

No experimento 3, um componente já desenvolvido para o controlador POX, chamado *flow_stats*, foi modificado para funcionar como uma ferramenta de segurança, baseando-se em valores de desvios padrões para identificar anomalias na rede, tendo como contramedidas a emissão de alertas em textos e bloqueios de portas do *switch* para isolar o *host* atacante. Além da identificação de anomalias, outro componente foi modificado (*host_tracker*), para auxiliar a identificação de intrusos na rede, juntamente uma máquina preparada para escanear a rede periodicamente, com um *script* implementado com a ferramenta *nmap*. Isso possibilita a identificação dos intrusos mesmo que estes, ainda, não tenham iniciado um ataque.

6 CONCLUSÃO

Por meio dos testes realizados, no primeiro experimento foi possível avaliar o comportamento da troca de pacotes DNP3 entre o mestre e o escravo em uma rede *Smart Grid*. Dessa forma, obtiveram-se resultados, em que, nos ataques direcionados para o mestre, a média de tempo na troca de pacotes DNP3 aumentou conforme o aumento da velocidade do fluxo, e, nos ataque direcionados para o escravo, a velocidade do fluxo não impactou o desempenho na troca de pacotes DNP3, mas sim o fator duração do ataque, que prejudicou o funcionamento do escravo até que parasse de responder, derrubando o escravo da rede *Smart Grid*.

O tempo que o mestre e o escravo levam para trocar pacotes pode ser crítico em um sistema real, pois o tempo em torno de um minuto pode significar dados irreparáveis em uma rede *Smart Grid*.

No segundo experimento, a troca de pacotes *OpenFlow* entre o *switch* e o controlador teve uma comunicação estável, mesmo aumentando o número de escravos da rede *Smart Grid*. Os picos na comunicação entre o controlador e *switch*, apresentados nas Figura 18 e 19, mostraram que, apenas no início da comunicação, o número de escravos influenciou a quantidade de pacotes *OpenFlow* entre o controlador e o *switch*. Todavia, a inicialização simultânea de todos os escravos em uma rede *Smart Grid* não é comum em um ambiente real. Sabendo-se da importância de medir o fluxo de dados entre o controlador e o *switch* para evitar um gargalo na rede SDN, os testes tiveram um resultado satisfatório, com duração máxima menor ou igual a 40 ms, mantendo uma boa performance da rede, de acordo com os valores de *delay* apresentados por Lu et al. (2011) (16 – 100 ms para monitoramento em tempo real).

Por fim, no terceiro experimento, com uma proposta de uma solução de segurança para *Smart Grid*, foram obtidas diferentes características em cada intervalo no momento dos ataques, como no momento da limpeza da tabela de fluxo e o fluxo no OVS quando atacado por máquinas invasoras e escravas da rede *Smart Grid.* Outro comportamento que chamou a atenção foi o atraso nas leituras da tabela de fluxo com o processamento do componente *host_tracker.* Esse comportamento foi consequência do intervalo de 10 segundos na leitura da tabela de fluxo, pois, com um tempo maior, esse atraso seria evitado. Outro ponto importante foi o desvio padrão encontrado, com valor pequeno, por consequência do fluxo contínuo, dando um intervalo de mínimo e máximo muito curto no

experimento. Porém, tanto o número de desvios padrões como o tempo de leitura da tabela de fluxo são parâmetros ajustáveis no arquivo de configurações.

Os resultados para os dois ataques no mestre da rede *Smart Grid* foram satisfatórios, com até 99% dos pacotes bloqueados pelo controlador e quase nenhuma anomalia no mestre.

Sabendo da importância em preservar a segurança em redes *Smart Grid*, o componente *flow_stats,* modificado do controlador POX, apresenta uma alternativa para complementar a integridade de uma rede *Smart Grid*, sendo capaz de monitorar não apenas o fluxo total do *switch*, mas de cada fluxo individualmente, além da arquitetura possuir um sensor para identificação de intrusos na rede e bloqueio de portas com novos fluxos no período de monitoramento.

Assim, pode-se concluir que uma política de segurança em uma rede *Smart Grid* deve ser item obrigatório para garantir uma proteção satisfatória, pois os danos causados por um monitoramento lento podem resultar em falhas. Além disso, não houve grandes impactos na comunicação com o aumento de escravos na rede *Smart Grid.* O segundo experimento mostrou uma comunicação adequada para que uma SDN seja capaz de monitorar uma *Smart Grid* de forma eficiente. Dessa forma, o monitoramento detalhado dos fluxos e mecanismos de contramedidas para impedir um ataque merece atenção como solução de segurança, pois mostrou-se eficiente nas condições apresentadas com o uso de desvios padrões, além de não ter sido encontrada uma arquitetura projetada com tal flexibilidade.

O próximo passo consiste em monitorar um número maior de escravos, a fim de analisar o comportamento de fluxo em vários *switches,* pois o número utilizado de escravos é adequado para uma aplicação no domínio do consumidor. Outros objetivos consistem em evitar o uso do componente *host_tracker*, adicionando sua função no componente *flow_stats* modificado e uma implementação de uma interface gráfica para configuração do componente e, por fim, criar uma interface gráfica para facilitar a configuração dos parâmetros.

REFERÊNCIAS

ABIDEEN, Z.; HASSAN, S. Enabling Smart Grids through advanced communication and control. In: IEEE INTERNATIONAL CONFERENCE ON SMART GRID AND SMART CITIES, 1., 2017, Singapore. **Proceedings...** Singapore: ICSGSC, 2017. p. 189-194.

APURVA, M.; HIMANSHU, K. Towards addressing common security issues in Smart Grid specifications. In: INTERNATIONAL SYMPOSIUM ON RESILIENT CONTROL SYSTEMS, 5., 2012, Salt Lake. **Proceedings...** Salt Lake: IEEE, 2012. p. 174-180.

AXON GROUP. **Axon test**. [S.l.: s.n.], 2017. Disponível em: <http://www.axongroup.com.co/productos/axon-test/>. Acesso em: 10 out 2017.

BAIG, Z. A.; AMOUDI, A. R. An analysis of smart grid attacks and countermeasures. **Journal of Communications**, Miami, v. 8, n. 8, p. 473-479, 2013.

CAHN, A. et al. Software-defined energy communication networks: From substation automation to future smart grids. In: IEEE INTERNATIONAL CONFERENCE ON SMART GRID COMMUNICATIONS (SMART GRIDCOMM), 10., 2013, Vancouver. **Proceedings...** Vancouver: IEEE, 2013. p. 558–563.

CAMARGO, C. **Smart Grid**: a rede elétrica inteligente. [S.l.: s.n.], 2009. Disponível em: <http://www.tecmundo.com.br/3008-smart-grid-a-rede-eletrica-inteligente.htm>. Acesso em: 3 abr 2013.

BUSSAB, W. O.; MORETTIN, P. A.. **Estatística básica**. Ed. 9, Saraiva, 2017.

DNP USERS GROUP. **Overview of the DNP3 protocol**. [S.l.: s.n.], 2017. Disponível em:<https://www.dnp.org/pages/aboutdefault.aspx>. Acesso em: 25 set. 2017.

DNP3. **Overview**. [S.l.: s.n.], 2013b. Disponível em: <https://github.com/gec/dnp3/blob/master/README>. Acesso em: 11 out. 2017.

DNP3. **Test set user manual**. [S.l.: s.n.], 2013a. Disponível em: <https://github.com/gec/dnp3/blob/master/docs/TestSetUserManual.docx>. Acesso em: 11 out 2017.

DONG, X. et al. Software-defined networking for Smart Grid resilience: Opportunities and challenges. In: ACM WORKSHOP ON CYBERPHYSICAL SYSTEM SECURITY, 1., 2015, Singapore. **Proceedings...** Singapore: ACM, 2015. p. 14–68.

FANG, X. et al. Smart grid – the new and improved power grid: a survey. **IEEE Communications Surveys & Tutorials**, Piscataway, v. 19, n. 4, p. 944-980, 2012.

FU, C.; NI, Z. The application of embedded system in Supervisory Control and Data Acquisition System (SCADA) over wireless sensor and GPRS networks. In: IEEE INTERNATIONAL CONFERENCE ON ANTI-COUNTERFEITING, 9., 2015, Xiamen. **Proceedings...** Xiamen: IEEE, 2015. p. 81-85.

FULLER, J. C. et al. Communication simulations for power system applications. In: WORKSHOP ON MODELING AND SIMULATION OF CYBER-PHYSICAL ENERGY SYSTEMS- MSCPES, 2013, Berkeley. **Proceedings...** Berkeley: IEEE, 2013. p. 1–6.

GAUR, G. et al. Demand side management in a Smart Grid environment. In: IEEE INTERNATIONAL CONFERENCE ON SMART GRID AND SMART CITIES-ICSGSC, 1, 2017, Singapore. **Proceedings...** Singapore: IEEE, 2017. p. 227 – 231.

GELLINGS, C. et al. **Estimating the costs and benefits of the Smart Grid**: a preliminary estimate of the investment requirements and the resultant benefits of a fully functioning Smart Grid. Palo Alto: Electric Power Research Institute, 2011. p. 5-9.

GENI CONCEPTS. **Project**. [S.l.: s.n.], 2017. Disponível em: <http://www.geni.net/documentation/geni-concepts/#>. Acesso em: 6 out 2017.

GENI PROJECT. **GENI**. [S.l.: s.n.], 2017. Disponível em: <http://www.geni.net/>. Acesso em: 6 out 2017.

GENI. **Hello GENI**. [S.l.: s.n.], 2017. Disponível em: <http://groups.geni.net/geni/wiki/GENIExperimenter/Tutorials/RunHelloGENI>. Acesso em: 05 dez 2017.

HITTINI, H.; ABDRABOU, A.; ZHANG, L. Sadsa: security aware distribution system architecture for smart grid applications. In: INTERNATIONAL CONFERENCE ON INNOVATIONS IN INFORMATION TECHNOLOGY, 12., 2016, Al Ain. **Proceedings...** Al Ain: IEEE, 2016. p. 1- 6.

INTERNATIONAL COMPUTER SCIENCE INSTITUTE. **NOX network control platform**. [S.l.: s.n.], 2014. Disponível em: <https://github.com/noxrepo/nox.>. Acesso em: 28 set 2017.

JACOBSON, V.; LERES, C.; MCCANNE, S. **TCPDUMP**. [S.l.: s.n.], 2017. Disponível em: <http://www.tcpdump.org/tcpdump_man.html>. Acesso em: 11 out. 2017.

JIN, D.; NICOL, D. M.; YAN, G. An event buffer flooding attack in DNP3 controlled SCADA systems. In: WINTER SIMULATION CONFERENCE- WSC, 11., 2011, Fenix. **Proceedings...** Fenix: Institute of Industrial Engineers, 2011. p. 2614 - 2626.

KAUR, S.; SINGH, J.; GHUMMAN, N. S. **Network programmability using POX controller**. Ferozepur: Department of Computer Science and Engineering, SBS State Technical Campus, 2014.

KIM, J.; FILALI, F.; Ko, Y. B. Trends and potentials of the Smart Grid infrastructure: from ICT sub-system to SDN-Enabled Smart Grid architecture. **Appl. Sci**., Basel, v. 5, n. 4, p. 706-727, 2015.

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

LOCKE, G.; GALLAGHER, P. D. **Nist framework and roadmap for Smart Grid interoperability standards, release 1.0**. Gaithersburg: Nat. Inst. Stand. Technol., 2010.

LONI, A.; PARAND, F. A. A survey of game theory approach in Smart Grid with emphasis on cooperative games. In: IEEE INTERNATIONAL CONFERENCE ON SMART GRID AND SMART CITIES, 1., 2017, Singapore. **Proceedings...** Singapore: IEEE, 2017. p. 237 – 242.

LOPES, Y. et al. Desafios de Segurança e Confiabilidade na Comunicação para Smart Grids. In: SIMPÓSIO BRASILEIRO EM SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS – SBSeg, 16, 2016, Niterói.
Anais... Niterói: Universidade Federal Fluminense, 2016. p. 142-186.

LU, X. et al. On network performance evaluation toward the smart grid: a case studyof DNP3 over TCP/IP. In: GLOBAL TELECOMMUNICATIONS CONFERENCE- GLOBECOM, 6., 2011, Kathmandu. **Proceedings...** Kathmandu: IEEE, 2011. p. 5-9.

MCCAULEY, J. **POX is a networking software platform written in Python**. [S.l.:s.n.], 2013. Disponível em: <https://github.com/noxrepo/pox>. Acesso em: 29 set. 2017.

MOLINA, E. et al. Using software defined networking to manage and control IEC 61850-based systems. **Comput. Electr. Eng**., Amsterdam, v. 43, n. 3, p. 142–154, 2015.

MOMOH, J. **Smart Grid architectural designs, Smart Grid**: fundamentals of design and analysis. Vancouver: Wiley-IEEE Press e Book Chapters, 2012. p. 1-15.

OH, H.; LEE, J.; KIM, C. A flow-based hybrid mechanism to improve performance in NOX and wireless openflow switch networks. In: IEEE VEHICULAR TECHNOLOGY CONFERENCE- VTC Fall, 74., 2011, San Francisco. **Proceedings...** San Francisco: IEEE, 2011. p. 1 – 4.

OPEN VSWITCH. **What is open vswitch?** [S.l.:s.n.], 2017. Disponível em: <http://docs.openvswitch.org/en/latest/intro/what-is-ovs/>. Acesso em: 11 out 2017.

ORTEGA, A. et al. **Proposal DNP3 protocol simulation on NS-2 in IEEE 802.11g wireless network ad hoc over TCP/IP in Smart Grid applications**. Quito: IEEE PES Innovative Smart Grid Technologies Latin America-ISGT LATAM, 2015a. p. 635-640.

ORTEGA, A. Análise de desempenho de redes de comunicação Wireless em aplicações de smart grid. 2015b. 128 f. Tese (Doutorado) – Universidade Estadual Paulista, Ilha Solteira, 2015b.

QIN, Z. et al. A software defined networking architecture for the internet-ofthings. In: IEEE NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM- NOMS, 27., 2014, Hsinchu. **Proceedings...** Hsinchu: IEEE, 2014. p. 1-9.

RINALDI, S. et al. Software defined networking applied to the heterogeneous infrastructure of Smart Grid. In: IEEE WORLD CONFERENCE ON FACTORY COMMUNICATION SYSTEMS- WFCS, 11., 2015, Palma de Mallorca. **Proceedings...** Palma de Mallorca: IEEE, 2015. p.1-4.

RODRIGUES, C. P. et al. Avaliação de balanceamento de carga web em redes definidas por software. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS– SBRC, 33., 2015, Vitória. **Anais...** Vitória: Universidade Federal do Espírito Santo, 2015. p. 18-22.

SILVA, F. O. Endereçamento por título: uma forma de encaminhamento multicast para a próxima geração de redes de computadores. 2013. 184 f. Tese (Doutorado) – Escola Politécnica, Universidade de São Paulo, São Paulo, 2013.

TAN, S. et al. Survey of security advances in Smart Grid: a data driven approach. **IEEE Communications Surveys & Tutorials**, Hsinchu, v. 19, n. 2, p. 397-422, 2017.

TCPREPLAY. **Welcome to tcpreplay**. [S.l.:s.n.], 2017. Disponível em: <http://tcpreplay.synfin.net/>. Acesso em: 10 out 2017.

VIRTUALBOX. **Features overview**. [S.l.:s.n.], 2017a. Disponível em: <https://www.virtualbox.org/manual/ch01.html#features-overview>. Acesso em: 9 out 2017a.

VIRTUALBOX. **Welcome to virtualbox.org!**. [S.l.:s.n.], 2017b. Disponível em: <https://www.virtualbox.org/>. Acesso em: 9 out 2017.

WHAT IS GENI. **What is GENI**. [S.l.:s.n.], 2017. Disponível em: <http://www.geni.net/about-geni/what-is-geni/>. Acesso em: 6 out 2017.

WIRESHARK. **About wireshark**. [S.l.:s.n.], 2017. Disponível em: <https://www.wireshark.org/>. Acesso em: 10 out. 2017.

XIA, W. et al. A Survey on software-defined networking. **IEEE Communications Surveys & Tutorials**, Piscataway, v. 17, p. 27-51, 2015.

YAN, Q. et al. Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) attacks in cloud computing environments: a survey, some research issues, and challenges. **IEEE Communications Surveys & Tutorials**, Piscataway, v. 18, p. 602-622, 2016.

ZHANG, J. et al. Opportunities for software-defined networking in Smart Grid. In: INTERNATIONAL CONFERENCE ON INFORMATION, COMMUNICATIONS AND SIGNAL PROCESSING- ICICS, 9., 2013, Tainan. **Proceedings...** Tainan: IEEXplore, 2013. p. 1-5.

ZHU, Y. et al. Joint substation-transmission line vulnerability assessment against the smart grid. **IEEE Transactions on Information Forensics and Security**, Piscataway, v. 10, n. 5, p. 1010-1024, 2015.

ANEXO 1 – PUBLICAÇÕES

FERRARI, RICARDO CESAR CÂMARA; SHINODA, AILTON AKIRA; SCHWEITZER, CHRISTIANE MARIE. Ataques TCPReplay em uma SmartGrid Emulada. In: Simpósio de Aplicações Operacionais em Áreas de Defesa, 2015, São José dos Campos, Brasil.

FERRARI, RICARDO CESAR CÂMARA; DE JESUS, RAFAEL MARCELINO; JARDIM JR, ELERSON GAETTI; SHINODA, AILTON AKIRA; SCHWEITZER, CHRISTIANE MARIE. ARQUITETURA PARA PROTEÇÃO DE UMA SMARTGRID CONTRA ATAQUES TCPREPLAY UTILIZANDO SDN. In: 2016 COPEC, 2016, Salvador, Brasil. XIV International Conference on Engineering and Technology Education, DOI 10.14684, INTERTECH 2016, 57-61.

DE JESUS, RAFAEL MARCELINO; FERRARI, RICARDO CESAR CÂMARA; JARDIM JR, ELERSON GAETTI; SHINODA, AILTON AKIRA; SCHWEITZER, CHRISTIANE MARIE. AVALIAÇÃO DOS CONTROLADORES BEACON E FLOODLIGHT, COM O USO DO EMULADOR MININET APLICADO A REDES DEFINIDAS POR SOFTWARE – SDN. In: 2016 COPEC, 2016, Salvador, Brasil. XIV International Conference on Engineering and Technology Education, DOI 10.14684 INTERTECH 2016, 51-55

FERRARI, RICARDO CESAR CÂMARA; SHINODA, AILTON AKIRA; SCHWEITZER, CHRISTIANE MARIE. Análise de tráfego entre OVS e controlador SDN. In: Congresso Nacional de Matemática Aplicada e Computacional (CNMAC), 2017, São José dos Campos, Brasil. Proceeding Series of the Brazilian Society of Computational and Applied Mathematics.

FERRARI, RICARDO CESAR CÂMARA; SHINODA, AILTON AKIRA; SCHWEITZER, CHRISTIANE MARIE. Controlador POX para a detecção de anomalias em redes Smart Grid. In: DINCOM 2017 - CONFERÊNCIA BRASILEIRA DE DINÂMICA, CONTROLE E APLICAÇÕES, 2017, São José do Rio Preto, Brasil.

FERRARI, RICARDO CESAR CÂMARA; SHINODA, AILTON AKIRA; SCHWEITZER, CHRISTIANE MARIE. Redes Definidas por Software para a Detecção de Anomalias e

Contra Medidas em Redes Smart Grid. In: 7º Simpósio de Processamento de Sinais (SPS), 2017, São Bernardo do Campo, Brasil.

FERRARI, RICARDO CESAR CÂMARA; SHINODA, AILTON AKIRA; SCHWEITZER, CHRISTIANE MARIE. A Feasibility Study On The Network Application Of Smart Grid Software For The Detection Of Anomalies And Countermeasures In Smart Grid Networks. Submetido para a revista *Computers & Security – Elsevier.*

ANEXO 2 – COMPONENTE *FLOW_STATS* MODIFICADO

```
# standard includes

from pox.lib.addresses import IPAddr, EthAddr
from pox.core import core
from pox.lib.util import dpidToStr
from math import sqrt
from datetime import datetime
import pox.openflow.libopenflow_01 as of
import os
import commands
import sys
import time

# include as part of the betta branch
from pox.openflow.of_json import *

log = core.getLogger()

#abre o arquivo de configuracao
f=open('/users/ricardof/smartgrid.config', 'r')

#pega as linhas do arquivo de configuracao
linhas=f.readlines()

#sera executada a leitura da estatisca de x em x seg.
leitura=int(linhas[0])

#limite de valores de diferencas de pacotes e bytes para determinar o conjunto de valores para a media e desvio
limite=int(linhas[1])

#aceita fluxo quando for verdadeira
```

```
aceita_fluxo = True

#ip do computador que ira trabalhar como sensor da rede (scan)
ip_sensor=str(linhas[2])

#ip do honeypot da rede, para receber ataques
ip_honey=str(linhas[4])

#numero de slaves na rede smartgrid
slaves=int(linhas[3])

#limite de valores de desvios padrao para fluxo
lim_desvios=int(linhas[5])

#limite de valores de desvios padrao para switch
lim_desvios_switch=int(linhas[6])

#ip do mestre da rede
ip_mestre=str(linhas[7])

#porta do mestre da rede
porta_mestre=str(linhas[8])

#tempo de bloqueio da porta
bloqueio_porta=int(linhas[9])

#numero de multiplicacao do desvio
ndesvio=int(linhas[10])

#se for verdadeiro esta aguardando escaneamento
primeira_vez_sensor = True

#armazena o protocolo e ip da vitima
vitimas_conhecidas=""
```

```
#contador de intervalos, para atribuir um intervalor para cada desvio
intervalos = {}

#aceita bloqueio da porta e depois recusa para nao bloquear a vitima
aceita_bloqueio=True

#nao aceita bloqueio de ip conhecido na primeira vez, para pega o atacante, pois esta bloqueando primeiro a vitima
aceita_bloqueio_conhecido=False

#tempo que a porta bloqueado vai ficar ate ser desbloqueada
tempo_bloqueio=-1

#pega o IP do host origem conhecido pela rede que esta atacando
host_conhecido_origem=""

#armazena o ip da vitima para retirar do vetor de portas bloqueadas
ip_vitima=0

#pega o IP do host destino conhecido pela rede que esta atacando
host_conhecido_destino=""

#determina se a tabela de fluxo sera atualizada ou nao
atualiza_tabela=False

#conta o total de chamadas dos fluxos, ex: 4 fluxos * limite, se o limite = 20, entao 80 chamadas no total.
#Isso para colocar aceita_fluxo = False e bloquear algum intruso depois que os devios foram obtidos
#total de chamada sera comparado com o total anterior, se for igual eh pq todos fluxos ja calcularam seus desvios
total_chamada=1
total_chamada_anterior=1
```

```
#vetores para armazenar os valores anteriores de cada fluxo
fluxo_anterior_pacotes= {}
fluxo_anterior_bytes= {}

#vetor para armazenar o valor de chamadas para cada fluxo individual
chamadas={}

#vetor para armazenar o valor da porta do switch de cada maquina com o ips como identificador
portas={}

#vetor para armazenar o valor da porta do switch de cada maquina com o ips como identificador
portas_bloqueadas={}

#vetor para armazenar os atacantes
atacantes={}

#armazena a hora atual
now=0

#variavel para armazenar as chamadas do switch para calculo do desvio padrao
chamadas_switch=0

#vetor para armazenar o valor de chamadas para cada fluxo individual depois que ja foram definidos os desvios
chamadas_analise={}

#vetores para armazenar os valores dos desvios minimo e maximo para cada fluxo individual
desvio_minimop={}
desvio_maximop={}
desvio_minimob={}
```

```
desvio_maximob={}

#vetor para armazenar a quantidade de desvios calculados para cada fluxo individual
num_desvios={}

#vetor para armazenar a quantidade de desvios calculados para cada switch
switch_desvios=1

#armazena o PDID switch
id_switch=0

#conta chamada do switch apenas uma vez, para nao contar a quantidade de fluxos vezes a chamada
conta_chamada=True

#vetor que armazena os valores das diferencas de cada fluxo
fluxo={}

#vetor que armazena os valores das diferencas de cada switch
switch={}

#armazena os totais de pacotes e bytes do switch
total_diferenca_pacotes=0
total_diferenca_bytes=0

#armazena os minimos e maximos do switch
switch_pacote_min=0
switch_pacote_max=0
switch_byte_min=0
switch_byte_max=0

# handler for timer function that sends the requests to all the
# switches connected to the controller.
def _timer_func ():
```

```
  for connection in core.openflow._connections.values():
    connection.send(of.ofp_stats_request(body=of.ofp_flow_stats_request()))
    connection.send(of.ofp_stats_request(body=of.ofp_port_stats_request()))
  log.debug("Sent %i flow/port stats request(s)", len(core.openflow._connections))

# handler to display flow statistics received in JSON format
# structure of event.stats is defined by ofp_flow_stats()
def _handle_flowstats_received (event):
  stats = flow_stats_to_list(event.stats)
  log.debug("FlowStatsReceived from %s: %s",
    dpidToStr(event.connection.dpid), stats)

  # Get number of bytes/packets in flows for web traffic only
  web_bytes = 0
  web_flows = 0
  web_packet = 0
  ips = 0
  ipd = 0
  porta = 0
  bytesporpacote = 0
  media_bytes = 0
  total_bytes = 0
  total_pacotes = 0
  protocolo = 0
  x=int(commands.getoutput("cat ~/chamadas"))
  palavra=''
  delta_pacotes=0
  ch=0
  somap=0
  somab=0
  somavarp=0
  somavarb=0

  #variaveis globais
```

```
  global fluxo_anterior_pacotes, fluxo_anterior_bytes, tamanho, chamadas, fluxo, limite, sensor, aceita_fluxo, total_chamada_anterior, total_chamada, primeira_vez_sensor, num_desvios, lim_desvios, lim_desvios_switch, intervalos, chamadas_analise, switch, chamadas_switch, switch_desvios, id_switch, total_diferenca_pacotes, total_diferenca_bytes, switch_pacote_min, switch_pacote_max, switch_byte_min, switch_byte_max, conta_chamada, now, aceita_bloqueio, tempo_bloqueio, host_conhecido_origem, host_conhecido_destino, aceita_bloqueio_conhecido, vitimas_conhecidas, porta_mestre, ip_mestre, portas, portas_bloqueadas, atualiza_tabela, ip_vitima, bloqueio_porta, ndesvio

 #pega o ID do switch
 id_switch=dpidToStr(event.connection.dpid)

#pega a quantidade de bytes por fluxo, 2 escravos 4 fluxos, 4 totais de bytes

#tentar associar origem e destino para somar os fluxos
 x=x+1
 chamadas_switch=chamadas_switch+1

  #atribui as diferencas de pacote e bytes de cada chamada no vetor switch igual a zero. Se tiver fluxo e entrar no FOR o valor muda somando as diferencas de todos os fluxo, se nao fica com zero pq nao tem fluxo ativo no switch

 if (chamadas_switch<=limite):
   try:
     switch[id_switch]['p'][chamadas_switch]=0
     switch[id_switch]['b'][chamadas_switch]=0
   except:
     switch[id_switch]={}
     switch[id_switch]['p']={}
     switch[id_switch]['b']={}
     switch[id_switch]['p'][chamadas_switch]=0
     switch[id_switch]['b'][chamadas_switch]=0
```

```
  palavra=str(x)
  palavra="echo "+palavra+" > ~/chamadas"
  os.system(palavra)

  for f in event.stats:
    web_bytes += f.byte_count
    web_packet += f.packet_count
    web_flows += 1
    porta = f.match.in_port
    ipd = f.match.nw_dst
    ips = f.match.nw_src
    total_bytes += web_bytes
    total_pacotes += web_packet
    protocolo = f.match.nw_proto

#para isso a melhor opcao de honeypot eh com baixa interatividade, pois esse tipo apenas simula os servicos, nao permitindo que o atacante consiga obter controle da maquina honeypot
#assim consigo obter dados de quais tipos de ataques estao sendo utilizados na rede
#outra coisa importante eh definir onde ficara o honeypot (internet-dmz-rede), antes, dentro ou depois da dmz

      #se vier ou esta indo para o sensor, desconsidera totalmente o fluxo, nao contando nada
    if (ips==ip_sensor) or (ipd==ip_sensor):
      #retira os valores de fluxo para nao contar nada
      web_flows-=1
      web_bytes -= f.byte_count
      web_packet -= f.packet_count
      total_bytes -= web_bytes
      total_pacotes -= web_packet

      #se for a primeira vez que o switch recebe uma mensagem do sensor eh pq fez a scan da rede e define que todos os hosts foram encontrados
```

```
    if (primeira_vez_sensor):
        #cria o arquivo de scan final
      palavra="cat ~/ips_scan > ~/ips_scan_final"
      os.system(palavra)
        primeira_vez_sensor=False

   #se vier ou esta indo para o honeypot desconsiderar, ou seja, deixa atacar para colher informacoes
   elif (ips==ip_honey) or (ipd==ip_honey):
     #retira os valores de fluxo para nao contar nada
     web_flows-=1
     web_bytes -= f.byte_count
     web_packet -= f.packet_count
     total_bytes -= web_bytes
     total_pacotes -= web_packet
     log.info("O honeypot esta sendo atacado")

   else:

     try:
        id_fluxo=str(protocolo)+str(porta)+str(ips)+str(id_switch)+str(ipd)
     except:
        log.info("")

      #se nao tiver protocolo e nem porta limpa, pois estava dando problema com um fluxo que nao serve para nada
     if (protocolo==None) and (porta==None):
        event.connection.send( of.ofp_flow_mod(command=of.OFPFC_DELETE))

       #se nao tiver protocolo limpa, pois estava danda problema com um fluxo que nao serve para nada
     if (protocolo==None):
        event.connection.send( of.ofp_flow_mod(match=of.ofp_match(in_port=porta)))
```

```
#se nao existir a chave para o dicionario (vetor) cria = 0
try:
    chamadas[id_fluxo]=chamadas[id_fluxo]+1
except:
    chamadas[id_fluxo]=1
    portas[ips]=porta
    atacantes[ips]=0
    log.info("Criando chaves:%s para o vetor de fluxo", id_fluxo)

#se nao existir a chave para o desvio desse fluxo, cria
try:
    if (num_desvios[id_fluxo]==0):
      num_desvios[id_fluxo]=num_desvios[id_fluxo]+1
except:
    num_desvios[id_fluxo]=1

try:
    diferenca_pacotes=f.packet_count-fluxo_anterior_pacotes[id_fluxo]
except:
    diferenca_pacotes=f.packet_count-0
    log.info("Criando chaves:%s para o vetor de fluxo", id_fluxo)

try:
    diferenca_bytes=f.byte_count-fluxo_anterior_bytes[id_fluxo]
except:
    diferenca_bytes=f.byte_count-0
    log.info("Criando chaves:%s para o vetor de fluxo", id_fluxo)

if ((diferenca_pacotes < 0) or (diferenca_bytes < 0)):
    diferenca_pacotes = fluxo_anterior_pacotes[id_fluxo] + diferenca_pacotes
    diferenca_bytes = fluxo_anterior_bytes[id_fluxo] + diferenca_bytes

#soma pacotes e bytes dos fluxos
total_diferenca_pacotes=total_diferenca_pacotes+diferenca_pacotes
```

```
    total_diferenca_bytes=total_diferenca_bytes+diferenca_bytes

# obtem ip origem e destino

    host_conhecido_origem=commands.getoutput("cat ~/ips_scan_final | grep " + str(ips))
    host_conhecido_destino=commands.getoutput("cat ~/ips_scan_final | grep " + str(ipd))

     #essa condicao detecta o fluxo bloqueado que nao esta transmitindo nenhum pacote (se for host conhecido), e isso quer dizer que eh a porta da vitima, e exclui a porta vitima do vetor de portas que precisam ser bloqueadas. Isso eh possivel apenas com slaves que somente respondem o master, ou seja, se o slave enviar mensagens nao solicitadas as duas portas serao bloqueadas, pois nao da pra identificar o atacante com as duas maquinas enviando pacotes mesmo com as portas bloqueadas

    #para limpar a vitima, ela nao pode estar enviando mensagens nao solicitadas, pois se tiver, sempre tera fluxo de dados nas duas portas e dessa forma nao da para identificar o atacante. Mas se nao estiver enviando mensagens nao solicitadas a vitima ira para de responder e assim podemos identificar o atacante

     if (ips==None) and (ipd==None) and (protocolo==None) and (f.packet_count==0) and (portas_bloqueadas):
          for i in portas_bloqueadas:
            if (portas_bloqueadas[i] == porta):
              ip_vitima=i
          del (portas_bloqueadas[ip_vitima])
          atualiza_tabela=True
          log.info("DESBLOQUEANDO PORTA: %s ", str(porta))

    #se eh ataque entre hosts conhecidos da rede
          if (chamadas[id_fluxo]==1) and (aceita_fluxo==False) and (protocolo!=None) and (ip_mestre!=ips) and (porta_mestre!=porta) and (host_conhecido_origem!="") and (host_conhecido_destino!=""):

          portas_bloqueadas[ips]=porta
```

```
        atualiza_tabela=True
        log.info("POSSIVEL ATAQUE DE INTRUSAO, PORTA %s BLOQUEADA (%s - %s)", porta, id_fluxo, chamadas[id_fluxo])
        chamadas[id_fluxo]=0 #zera as chamadas para esse fluxo para nao correr risco de numero de chamadas atingir o limite, se atingir o limite nao bloqueia mais o ataque
        tempo_bloqueio=bloqueio_porta
        now = datetime.now()
        os.system("echo "+str(id_fluxo)+"-"+str(now.hour)+":"+str(now.minute)+":"+str(now.second)+">> ~/atacantes_conhecidos")

         #se eh ataque entre um host conhecido e um desconhecido. Define que o desconhecido eh o atacante
       elif (chamadas[id_fluxo]==1) and (aceita_fluxo==False) and (protocolo!=None) and (host_conhecido_origem=="") and (host_conhecido_destino!=""):

        portas_bloqueadas[ips]=porta
        log.info("POSSIVEL ATAQUE DE INTRUSAO, PORTA %s BLOQUEADA (%s - %s)", porta, id_fluxo, chamadas[id_fluxo])
        chamadas[id_fluxo]=0 #zera as chamadas para esse fluxo para nao correr risco de numero de chamadas atingir o limite, se atingir o limite nao bloqueia mais o ataque
        tempo_bloqueio=bloqueio_porta
        atualiza_tabela=True
        now = datetime.now()
        os.system("echo "+str(id_fluxo)+"-"+str(now.hour)+":"+str(now.minute)+":"+str(now.second)+">> ~/atacantes_desconhecidos")

    #valores das diferencas de pacotes e bytes para cada chamada
         if (chamadas[id_fluxo]<=limite) and (num_desvios[id_fluxo]<=lim_desvios) and (protocolo!=None):

        total_chamada=total_chamada+1

        #cria uma referencia para o fluxo, por exemplo, fluxo[13][p][1]=2, fluxo id 13, para pacotes, na posicao 1 guarda a diferenca de pacotes 2
```

```
        try:
          fluxo[id_fluxo]['p'][chamadas[id_fluxo]]=diferenca_pacotes
          fluxo[id_fluxo]['b'][chamadas[id_fluxo]]=diferenca_bytes
        except:
          fluxo[id_fluxo]={}
          fluxo[id_fluxo]['p']={}
          fluxo[id_fluxo]['b']={}
          fluxo[id_fluxo]['p'][chamadas[id_fluxo]]=diferenca_pacotes
          fluxo[id_fluxo]['b'][chamadas[id_fluxo]]=diferenca_bytes
          log.info("Criando chaves:%s para o vetor de fluxo", id_fluxo)

        #soma as diferencas de pacotes e bytes de todos os fluxos
        if (chamadas_switch<=limite):
            switch[id_switch]['p'][chamadas_switch]=switch[id_switch]['p'][chamadas_switch]
+diferenca_pacotes
            switch[id_switch]['b'][chamadas_switch]=switch[id_switch]['b'][chamadas_switch]
+diferenca_bytes

    #calcula a media de pacotes e bytes /leitura
       elif (chamadas[id_fluxo]==limite+1) and (num_desvios[id_fluxo]<=lim_desvios) and
(protocolo!=None):
        ch=1 #chamadas
        total_chamada=total_chamada+1
        for ch in range(1, limite+1):
          somap=float(somap+fluxo[id_fluxo]['p'][ch]) #soma as diferencas/l do fluxo
          somab=float(somab+fluxo[id_fluxo]['b'][ch])
            somavarp=float(somavarp+fluxo[id_fluxo]['p'][ch]**2) #soma as diferencas/l ao
quadrado
          somavarb=float(somavarb+fluxo[id_fluxo]['b'][ch]**2)

        # media de pacotes e bytes de cada fluxo
        fluxo[id_fluxo]['mp']=float(somap)/limite
        fluxo[id_fluxo]['mb']=float(somab)/limite
```

```
        #calculo da variancia de cada fluxo
        fluxo[id_fluxo]['varianciap']= (somavarp - ((somap**2)/limite)) / (limite-1)
        fluxo[id_fluxo]['varianciab']= (somavarb - ((somab**2)/limite)) / (limite-1)

        #calcula o devio padrao de cada fluxo
        fluxo[id_fluxo]['desviop']= sqrt(fluxo[id_fluxo]['varianciap'])
        fluxo[id_fluxo]['desviob']= sqrt(fluxo[id_fluxo]['varianciab'])

        log.info(" %s: %s",id_fluxo, fluxo[id_fluxo])

        #calculo do minimo e maximo para o desvio padrao de pacotes e bytes
        fluxo[id_fluxo]['desvio_minimop']=fluxo[id_fluxo]['mp']-(ndesvio*fluxo[id_fluxo]
['desviop'])
        fluxo[id_fluxo]['desvio_maximop']=fluxo[id_fluxo]['mp']+(ndesvio*fluxo[id_fluxo]
['desviop'])
        fluxo[id_fluxo]['desvio_minimob']=fluxo[id_fluxo]['mb']-(ndesvio*fluxo[id_fluxo]
['desviob'])
        fluxo[id_fluxo]['desvio_maximob']=fluxo[id_fluxo]['mb']+(ndesvio*fluxo[id_fluxo]
['desviob'])

        #se o desvio minimo for menor que zero pega apenas a parte positiva
        if (fluxo[id_fluxo]['desvio_minimob'] < 0):
          fluxo[id_fluxo]['desvio_minimob'] = 0
        if (fluxo[id_fluxo]['desvio_minimop'] < 0):
          fluxo[id_fluxo]['desvio_minimop'] = 0

        #armazena os valores minimo e maximo para cada fluxo
        try:
                                fluxo[id_fluxo][str(num_desvios[id_fluxo])]['minimop']=fluxo[id_fluxo]
['desvio_minimop']
                               fluxo[id_fluxo][str(num_desvios[id_fluxo])]['maximop']=fluxo[id_fluxo]
['desvio_maximop']
                                fluxo[id_fluxo][str(num_desvios[id_fluxo])]['minimob']=fluxo[id_fluxo]
['desvio_minimob']
```

```
                        fluxo[id_fluxo][str(num_desvios[id_fluxo])]['maximob']=fluxo[id_fluxo]
['desvio_maximob']
          #atribui o numero dos intervalos de cada desvio calculado, assim se tenho 5
desvios, tenho 5 intervalos, como se um dia nao tivesse 24 horas, mas 5 intervalos. Com
isso posso calcular o dia com 24 intervalos de 1 hora.
        except:
         fluxo[id_fluxo][str(num_desvios[id_fluxo])]={}
                         fluxo[id_fluxo][str(num_desvios[id_fluxo])]['minimop']=fluxo[id_fluxo]
['desvio_minimop']
                        fluxo[id_fluxo][str(num_desvios[id_fluxo])]['maximop']=fluxo[id_fluxo]
['desvio_maximop']
                         fluxo[id_fluxo][str(num_desvios[id_fluxo])]['minimob']=fluxo[id_fluxo]
['desvio_minimob']
                        fluxo[id_fluxo][str(num_desvios[id_fluxo])]['maximob']=fluxo[id_fluxo]
['desvio_maximob']

        #imprime no arquivo desvio os desvios de cada fluxo
        os.system("echo -----Resultado do desvio "+ str(num_desvios[id_fluxo]) +"----- >>
~/desvios")
        now = datetime.now()
      os.system("echo Hora:" + str(now.hour)+":"+str(now.minute)+":"+str(now.second)+" >>
~/desvios")
        os.system("echo minp:"+str(id_fluxo)+"--" +str(fluxo[id_fluxo]['desvio_minimop'])+"
>> ~/desvios; echo maxp:"+str(id_fluxo)+"--" +str(fluxo[id_fluxo]['desvio_maximop'])+" >>
~/desvios; echo minb:"+str(id_fluxo)+"--" +str(fluxo[id_fluxo]['desvio_minimob'])+" >>
~/desvios; echo maxb:"+str(id_fluxo)+"--" +str(fluxo[id_fluxo]['desvio_maximob'])+" >>
~/desvios")
        os.system("echo Conjunto - "+str(fluxo[id_fluxo])+" >> ~/desvios")
        os.system("echo . >> ~/desvios")

        somap=0
        somab=0
        somavarp=0
        somavarb=0
```

```
        #se o numero de desvios calculados for menor que o numero de desvio que precisam ser calculados, volta chamadas para 1, para comecar a calcular o proximo desvio
        if (num_desvios[id_fluxo]<lim_desvios):
          chamadas[id_fluxo]=1

        num_desvios[id_fluxo]=num_desvios[id_fluxo]+1

       # se a diferenca de pacotes ou bytes estiver fora do intervalo minimo e maximo do desvio acusa possivel ataque
      try:
        #se o numero de desvios for igual o limite de desvios eh pq ja foram feitos os calculos de todos os desvios, agora eh hora de iniciar as comparacoes do fluxo entre o minimo e maximo para detectar ataque

        #se acabou de calcular os desvios comeca a analisar os switches e troca de desvios
        if (switch_desvios>lim_desvios_switch) and (conta_chamada==True):
          #criar vetor para armazenar as chamadas de analise de cada switch

           #nao permite mais contar chamadas de analise, se nao entra aqui a quantidade de fluxos que tiver
          conta_chamada=False

          try:
            chamadas_analise[id_switch]=chamadas_analise[id_switch]+1
          except:
            chamadas_analise[id_switch]=1

            #recebe os valores de min a max para cada switch para analise
            try:
                switch[id_switch]['desvio_minimop']=switch[id_switch][str(intervalos[id_switch])]['minimop']
```

```
                switch[id_switch]['desvio_maximop']=switch[id_switch][str(intervalos[id_switch])]
['maximop']
                 switch[id_switch]['desvio_minimob']=switch[id_switch][str(intervalos[id_switch])]
['minimob']
                switch[id_switch]['desvio_maximob']=switch[id_switch][str(intervalos[id_switch])]
['maximob']
            except:
                intervalos[id_switch]=1
                 switch[id_switch]['desvio_minimop']=switch[id_switch][str(intervalos[id_switch])]
['minimop']
                switch[id_switch]['desvio_maximop']=switch[id_switch][str(intervalos[id_switch])]
['maximop']
                 switch[id_switch]['desvio_minimob']=switch[id_switch][str(intervalos[id_switch])]
['minimob']
                switch[id_switch]['desvio_maximob']=switch[id_switch][str(intervalos[id_switch])]
['maximob']

                os.system("echo Fora do IF >> ~/testesaida")
                os.system("echo Switch:"+ str(id_switch) + ">> ~/testesaida")
                os.system("echo intervalo:" + str(intervalos[id_switch]) +" >> ~/testesaida")
                now = datetime.now()
              os.system("echo Hora:" + str(now.hour)+":"+str(now.minute)+":"+str(now.second)+"
>> ~/testesaida")
                 os.system("echo chamadas_analise:" + str(chamadas_analise[id_switch]) +" >>
~/testesaida")
                 os.system("echo desvio_minimop:" + str(switch[id_switch]['desvio_minimop']) +"
>> ~/testesaida")
                  os.system("echo desvio_maximop:" + str(switch[id_switch]['desvio_maximop'])
+" >> ~/testesaida")
                 os.system("echo desvio_minimob:" + str(switch[id_switch]['desvio_minimob']) +"
>> ~/testesaida")
                  os.system("echo desvio_maximob:" + str(switch[id_switch]['desvio_maximob'])
+" >> ~/testesaida")
                os.system("echo . >> ~/testesaida")
```

```
        # se acabaram os desvios, nao aceita mais fluxo e chegou no numero de chamada de analise, troca os valores de desvios para os valores de desvios do proximo intervalo e volta as chamadas de analise

        #se o intervalo for igual ao limite desvio precisa voltar para 0
                        if ((intervalos[id_switch])==lim_desvios_switch) and (chamadas_analise[id_switch]==limite+1):
            intervalos[id_switch]=0

                                if ((intervalos[id_switch])<lim_desvios_switch) and (chamadas_analise[id_switch]==limite+1):
            #cria chave se nao existir
            intervalos[id_switch]=intervalos[id_switch]+1
            chamadas_analise[id_switch]=1
                switch[id_switch]['desvio_minimop']=switch[id_switch][str(intervalos[id_switch])]['minimop']
               switch[id_switch]['desvio_maximop']=switch[id_switch][str(intervalos[id_switch])]['maximop']
                switch[id_switch]['desvio_minimob']=switch[id_switch][str(intervalos[id_switch])]['minimob']
               switch[id_switch]['desvio_maximob']=switch[id_switch][str(intervalos[id_switch])]['maximob']

            os.system("echo Dentro do IF >> ~/testesaida")
            os.system("echo Switch:"+ str(id_switch) + ">> ~/testesaida")
            os.system("echo intervalo:" + str(intervalos[id_switch]) +" >> ~/testesaida")
            now = datetime.now()
           os.system("echo Hora:" + str(now.hour)+":"+str(now.minute)+":"+str(now.second)+" >> ~/testesaida")
             os.system("echo chamadas_analise:" + str(chamadas_analise[id_switch]) +" >> ~/testesaida")
             os.system("echo desvio_minimop:" + str(switch[id_switch]['desvio_minimop']) +" >> ~/testesaida")
```

```
            os.system("echo desvio_maximop:" + str(switch[id_switch]['desvio_maximop']) +"
>> ~/testesaida")
             os.system("echo desvio_minimob:" + str(switch[id_switch]['desvio_minimob']) +"
>> ~/testesaida")
            os.system("echo desvio_maximob:" + str(switch[id_switch]['desvio_maximob']) +"
>> ~/testesaida")
           os.system("echo . >> ~/testesaida")

        #atribui para variaveis globais para verificar fora do for, pois dentro verifica com
valores do fluxo e nao do total
        switch_pacote_min=switch[id_switch]['desvio_minimop']
        switch_pacote_max=switch[id_switch]['desvio_maximop']
        switch_byte_min=switch[id_switch]['desvio_minimob']
        switch_byte_max=switch[id_switch]['desvio_maximob']

        #se acabou de calcular os desvios comeca a analisar o fluxo e troca de desvios
        if (num_desvios[id_fluxo]>lim_desvios):
          #criar vetor para armazenar as chamadas de analise de cada fluxo

          try:
            chamadas_analise[id_fluxo]=chamadas_analise[id_fluxo]+1
          except:
            chamadas_analise[id_fluxo]=1

            #recebe os valores de min a max para cada fluxo para analise
            try:
                        fluxo[id_fluxo]['desvio_minimop']=fluxo[id_fluxo][str(intervalos[id_fluxo])]
['minimop']
                       fluxo[id_fluxo]['desvio_maximop']=fluxo[id_fluxo][str(intervalos[id_fluxo])]
['maximop']
                        fluxo[id_fluxo]['desvio_minimob']=fluxo[id_fluxo][str(intervalos[id_fluxo])]
['minimob']
                       fluxo[id_fluxo]['desvio_maximob']=fluxo[id_fluxo][str(intervalos[id_fluxo])]
['maximob']
```

```
        except:
            intervalos[id_fluxo]=1
                        fluxo[id_fluxo]['desvio_minimop']=fluxo[id_fluxo][str(intervalos[id_fluxo])]
['minimop']
                       fluxo[id_fluxo]['desvio_maximop']=fluxo[id_fluxo][str(intervalos[id_fluxo])]
['maximop']
                        fluxo[id_fluxo]['desvio_minimob']=fluxo[id_fluxo][str(intervalos[id_fluxo])]
['minimob']
                       fluxo[id_fluxo]['desvio_maximob']=fluxo[id_fluxo][str(intervalos[id_fluxo])]
['maximob']

            os.system("echo Fora do IF >> ~/testesaida")
            os.system("echo Fluxo:"+ str(id_fluxo) + ">> ~/testesaida")
            os.system("echo intervalo:" + str(intervalos[id_fluxo]) +" >> ~/testesaida")
            now = datetime.now()
          os.system("echo Hora:" + str(now.hour)+":"+str(now.minute)+":"+str(now.second)+"
>> ~/testesaida")
             os.system("echo chamadas_analise:" + str(chamadas_analise[id_fluxo]) +" >>
~/testesaida")
            os.system("echo desvio_minimop:" + str(fluxo[id_fluxo]['desvio_minimop']) +" >>
~/testesaida")
             os.system("echo desvio_maximop:" + str(fluxo[id_fluxo]['desvio_maximop']) +"
>> ~/testesaida")
            os.system("echo desvio_minimob:" + str(fluxo[id_fluxo]['desvio_minimob']) +" >>
~/testesaida")
             os.system("echo desvio_maximob:" + str(fluxo[id_fluxo]['desvio_maximob']) +"
>> ~/testesaida")
            os.system("echo . >> ~/testesaida")

         # se acabou os desvios, nao aceita mais fluxo e chegou no numero de chamada
de analise, troca os valores de desvios para os valores de desvios do proximo intervalo e
volta as chamadas de analise

        #se o intervalo for igual ao limite desvio precisa voltar para 0
```

```
                    if ((intervalos[id_fluxo])==lim_desvios) and
(chamadas_analise[id_fluxo]==limite+1):
        intervalos[id_fluxo]=0

    if ((intervalos[id_fluxo])<lim_desvios) and (chamadas_analise[id_fluxo]==limite+1):
        #cria chave se nao existir
        intervalos[id_fluxo]=intervalos[id_fluxo]+1
        chamadas_analise[id_fluxo]=1
        fluxo[id_fluxo]['desvio_minimop']=fluxo[id_fluxo][str(intervalos[id_fluxo])]['minimop']
                fluxo[id_fluxo]['desvio_maximop']=fluxo[id_fluxo][str(intervalos[id_fluxo])]
['maximop']
        fluxo[id_fluxo]['desvio_minimob']=fluxo[id_fluxo][str(intervalos[id_fluxo])]['minimob']
                fluxo[id_fluxo]['desvio_maximob']=fluxo[id_fluxo][str(intervalos[id_fluxo])]
['maximob']

        os.system("echo Dentro do IF >> ~/testesaida")
        os.system("echo Fluxo:"+ str(id_fluxo) + ">> ~/testesaida")
        os.system("echo intervalo:" + str(intervalos[id_fluxo]) +" >> ~/testesaida")
        now = datetime.now()
       os.system("echo Hora:" + str(now.hour)+":"+str(now.minute)+":"+str(now.second)+"
>> ~/testesaida")
         os.system("echo chamadas_analise:" + str(chamadas_analise[id_fluxo]) +" >>
~/testesaida")
         os.system("echo desvio_minimop:" + str(fluxo[id_fluxo]['desvio_minimop']) +" >>
~/testesaida")
        os.system("echo desvio_maximop:" + str(fluxo[id_fluxo]['desvio_maximop']) +" >>
~/testesaida")
         os.system("echo desvio_minimob:" + str(fluxo[id_fluxo]['desvio_minimob']) +" >>
~/testesaida")
        os.system("echo desvio_maximob:" + str(fluxo[id_fluxo]['desvio_maximob']) +" >>
~/testesaida")
        os.system("echo . >> ~/testesaida")
```

```
        if ((diferenca_pacotes < fluxo[id_fluxo]['desvio_minimop']) or (diferenca_pacotes > fluxo[id_fluxo]['desvio_maximop']) or (diferenca_bytes < fluxo[id_fluxo]['desvio_minimob']) or (diferenca_bytes > fluxo[id_fluxo]['desvio_maximob'])) and (num_desvios[id_fluxo]>lim_desvios):
            log.info("IDENTIFICANDO POSSIVEL ANOMALIA NO FLUXO! %s: Minp:%s e Maxp:%s | Minb:%s e Maxb:%s", id_fluxo, fluxo[id_fluxo]['desvio_minimop'], fluxo[id_fluxo]['desvio_maximop'], fluxo[id_fluxo]['desvio_minimob'], fluxo[id_fluxo]['desvio_maximob'])

            os.system("echo intervalo:" + str(intervalos[id_fluxo]) +" >> ~/falso_positivo")
            now = datetime.now()
            os.system("echo Hora:" + str(now.hour)+":"+str(now.minute)+":"+str(now.second)+" >> ~/falso_positivo")
            os.system("echo Fluxo:"+ str(id_fluxo) + ">> ~/falso_positivo")
                os.system("echo MinP: " + str(fluxo[id_fluxo]['desvio_minimop']) + " >> ~/falso_positivo")
                os.system("echo MaxP: " + str(fluxo[id_fluxo]['desvio_maximop']) + " >> ~/falso_positivo")
                os.system("echo MinB: " + str(fluxo[id_fluxo]['desvio_minimob']) + " >> ~/falso_positivo")
                os.system("echo MaxB: " + str(fluxo[id_fluxo]['desvio_maximob']) + " >> ~/falso_positivo")
            os.system("echo Pacotes: " + str(diferenca_pacotes) + " >> ~/falso_positivo")
            os.system("echo Bytes: " + str(diferenca_bytes) + " >> ~/falso_positivo")
            os.system("echo Fluxos: " + str(web_flows) + " >> ~/falso_positivo")
            os.system("echo Aceita Fluxo: " + str(aceita_fluxo) + " >> ~/falso_positivo")
            os.system("echo ---------- >> ~/falso_positivo")

        #armazena todos os valores coletados das chamadas no arquivo, para criar o grafico
        if (num_desvios[id_fluxo]>lim_desvios):
            now = datetime.now()
                os.system("echo "+str(id_fluxo)+" Desvio:"+str(intervalos[id_fluxo])+" Pacotes:"+str(diferenca_pacotes)+" "+ str(now.hour)+":"+str(now.minute)+":"+str(now.second)+">> ~/analise_pacotes")
```

```
                os.system("echo "+str(id_fluxo)+" Desvio:"+str(intervalos[id_fluxo])+" Bytes:"+str(diferenca_bytes)+" "+ str(now.hour)+":"+str(now.minute)+":"+str(now.second)+" >> ~/analise_bytes")

    except:
        if (aceita_fluxo==False):
          log.info("POSSIVEL ATAQUE DE INTRUSAO NA PORTA %s DO SWITCH.", porta)
        else:
         log.info("Aguarde, calculando desvio padrao para cada fluxo")

    fluxo_anterior_pacotes[id_fluxo] = f.packet_count
    fluxo_anterior_bytes[id_fluxo] = f.byte_count
     log.info(" %s --> %s - Bytes:%s - pacotes:%s - Proto:%s - porta:%s - Dif. Pacotes:%s - Dif. Bytes:%s - Chamada:%s", ips, ipd, f.byte_count, f.packet_count, protocolo, porta, diferenca_pacotes, diferenca_bytes, chamadas[id_fluxo])

   #se total_chamada==total_chamada_anterior eh pq nao entrou nenhuma vez para calcular o devio, isso quer dizer que os desvios ja estao calculados.
  if (total_chamada_anterior!=total_chamada) or (web_flows==0):
    total_chamada_anterior=total_chamada

   #se calculou todos os desvios do switch para de aceitar NOVOS fluxos, mas os que comecaram antes de calcular o ultimo desvio do switch terminam
  if (switch_desvios>lim_desvios_switch) and (aceita_fluxo):
    aceita_fluxo=False
    #define os IPS que participam da rede
    palavra="cat ~/ips_scan > ~/ips_scan_final"
    os.system(palavra)

  #calcular a desvios de pacotes e bytes do switch
  if (chamadas_switch==limite+1) and (switch_desvios<=lim_desvios_switch):
    chs=1
    for chs in range(1, limite+1):
```

```
    somap=float(somap+switch[id_switch]['p'][chs]) #soma as diferencas/leitura do fluxo
    somab=float(somab+switch[id_switch]['b'][chs])
      somavarp=float(somavarp+switch[id_switch]['p'][chs]**2) #soma as diferencas/leitura
ao quadrado
    somavarb=float(somavarb+switch[id_switch]['b'][chs]**2)

  # media de pacotes e bytes do switch
  switch[id_switch]['mp']=float(somap)/limite
  switch[id_switch]['mb']=float(somab)/limite

  #calculo da variancia do switch
  switch[id_switch]['varianciap']= (somavarp - ((somap**2)/limite)) / (limite-1)
  switch[id_switch]['varianciab']= (somavarb - ((somab**2)/limite)) / (limite-1)

  #calcula o devio padrao do switch
  switch[id_switch]['desviop']= sqrt(switch[id_switch]['varianciap'])
  switch[id_switch]['desviob']= sqrt(switch[id_switch]['varianciab'])

  #calculo do minimo e maximo para o desvio padrao de pacotes e bytes
      switch[id_switch]['desvio_minimop']=switch[id_switch]['mp']-(ndesvio*switch[id_switch]
['desviop'])
    switch[id_switch]['desvio_maximop']=switch[id_switch]['mp']+(ndesvio*switch[id_switch]
['desviop'])
      switch[id_switch]['desvio_minimob']=switch[id_switch]['mb']-(ndesvio*switch[id_switch]
['desviob'])
    switch[id_switch]['desvio_maximob']=switch[id_switch]['mb']+(ndesvio*switch[id_switch]
['desviob'])

  #se o desvio minimo for menor que zero pega apenas a parte positiva
  if (switch[id_switch]['desvio_minimob'] < 0):
    switch[id_switch]['desvio_minimob'] = 0
  if (switch[id_switch]['desvio_minimop'] < 0):
    switch[id_switch]['desvio_minimop'] = 0
```

```
    try:
     switch[id_switch][str(switch_desvios)]['minimop']=switch[id_switch]['desvio_minimop']
     switch[id_switch][str(switch_desvios)]['maximop']=switch[id_switch]['desvio_maximop']
     switch[id_switch][str(switch_desvios)]['minimob']=switch[id_switch]['desvio_minimob']
     switch[id_switch][str(switch_desvios)]['maximob']=switch[id_switch]['desvio_maximob']
      #atribui o numero dos intervalos de cada desvio calculado, assim se tenho 5 desvios, tenho 5 intervalos, como se um dia nao tivesse 24 horas, mas 5 intervalos. Com isso posso calcular o dia com 24 intervalos de 1 hora.
    except:
     switch[id_switch][str(switch_desvios)]={}
     switch[id_switch][str(switch_desvios)]['minimop']=switch[id_switch]['desvio_minimop']
     switch[id_switch][str(switch_desvios)]['maximop']=switch[id_switch]['desvio_maximop']
     switch[id_switch][str(switch_desvios)]['minimob']=switch[id_switch]['desvio_minimob']
     switch[id_switch][str(switch_desvios)]['maximob']=switch[id_switch]['desvio_maximob']

    #imprime no arquivo desvio os desvios do switch
      log.info("CONJUNTO SWITCH - P:%s | B:%s", switch[id_switch]['p'], switch[id_switch]['b'])
    now = datetime.now()
    os.system("echo -----DESVIO DO SWITCH "+ str(id_switch) +"----- >> ~/desvios")
      os.system("echo Hora:" + str(now.hour)+":"+str(now.minute)+":"+str(now.second)+" >> ~/desvios")
                os.system("echo minp:"+str(switch_desvios)+"--" +str(switch[id_switch]['desvio_minimop'])+" >> ~/desvios; echo maxp:"+str(switch_desvios)+"--" +str(switch[id_switch]['desvio_maximop'])+" >> ~/desvios; echo minb:"+str(switch_desvios)+"--" +str(switch[id_switch]['desvio_minimob'])+" >> ~/desvios; echo maxb:"+str(switch_desvios)+"--" +str(switch[id_switch]['desvio_maximob'])+" >> ~/desvios")
    os.system("echo Conjunto - "+str(switch[id_switch])+" >> ~/desvios")
    os.system("echo . >> ~/desvios")

    somap=0
    somab=0
    somavarp=0
    somavarb=0
```

```
    #se o numero de desvios calculados for menor que o numero de desvio que precisam ser calculados, volta chamadas para 1, para comecar a calcular o proximo desvio. Pega os valores da chamada que faz o calculos para nao perder
    if (switch_desvios<lim_desvios_switch):
      chamadas_switch=1
      switch[id_switch]['p'][chamadas_switch]=total_diferenca_pacotes
      switch[id_switch]['b'][chamadas_switch]=total_diferenca_bytes

    switch_desvios=switch_desvios+1

   #se acabou de calcular os desvios comeca a analisar os switches e troca de desvios, ESSA PARTE EXECUTA APENAS QUANDO NAO HOUVER FLUXO PARA ENTRAR NO FOR
   if (switch_desvios>lim_desvios_switch) and (conta_chamada==True):
    #criar vetor para armazenar as chamadas de analise de cada switch

     #nao permite mais contar chamadas de analise, se nao entra aqui a quantidade de fluxos que tiver
    conta_chamada=False

    try:
      chamadas_analise[id_switch]=chamadas_analise[id_switch]+1
    except:
      chamadas_analise[id_switch]=1

      #recebe os valores de min a max para cada switch para analise
      try:
          switch[id_switch]['desvio_minimop']=switch[id_switch][str(intervalos[id_switch])]['minimop']
          switch[id_switch]['desvio_maximop']=switch[id_switch][str(intervalos[id_switch])]['maximop']
          switch[id_switch]['desvio_minimob']=switch[id_switch][str(intervalos[id_switch])]['minimob']
```

```
        switch[id_switch]['desvio_maximob']=switch[id_switch][str(intervalos[id_switch])]
['maximob']
    except:
        intervalos[id_switch]=1
        switch[id_switch]['desvio_minimop']=switch[id_switch][str(intervalos[id_switch])]
['minimop']
        switch[id_switch]['desvio_maximop']=switch[id_switch][str(intervalos[id_switch])]
['maximop']
        switch[id_switch]['desvio_minimob']=switch[id_switch][str(intervalos[id_switch])]
['minimob']
        switch[id_switch]['desvio_maximob']=switch[id_switch][str(intervalos[id_switch])]
['maximob']

        os.system("echo AQUI 1 >> ~/testesaida")
        os.system("echo Switch:"+ str(id_switch) + ">> ~/testesaida")
        os.system("echo intervalo:" + str(intervalos[id_switch]) +" >> ~/testesaida")
        now = datetime.now()
      os.system("echo Hora:" + str(now.hour)+":"+str(now.minute)+":"+str(now.second)+" >>
~/testesaida")
        os.system("echo chamadas_analise:" + str(chamadas_analise[id_switch]) +" >>
~/testesaida")
        os.system("echo desvio_minimop:" + str(switch[id_switch]['desvio_minimop']) +" >>
~/testesaida")
        os.system("echo desvio_maximop:" + str(switch[id_switch]['desvio_maximop']) +" >>
~/testesaida")
        os.system("echo desvio_minimob:" + str(switch[id_switch]['desvio_minimob']) +" >>
~/testesaida")
        os.system("echo desvio_maximob:" + str(switch[id_switch]['desvio_maximob']) +" >>
~/testesaida")
        os.system("echo . >> ~/testesaida")

        # se acaboram os desvios, nao aceita mais fluxo e chegou no numero de chamada
de analise, troca os valores de desvios para os valores de desvios do proximo intervalo e
volta as chamadas de analise
```

```
        #se o intervalo for igual ao limite desvio precisa voltar para 0
                                if          ((intervalos[id_switch])==lim_desvios_switch)          and
(chamadas_analise[id_switch]==limite+1):
    intervalos[id_switch]=0

                                if          ((intervalos[id_switch])<lim_desvios_switch)          and
(chamadas_analise[id_switch]==limite+1):
    #cria chave se nao existir
    intervalos[id_switch]=intervalos[id_switch]+1
    chamadas_analise[id_switch]=1
             switch[id_switch]['desvio_minimop']=switch[id_switch][str(intervalos[id_switch])]
['minimop']
            switch[id_switch]['desvio_maximop']=switch[id_switch][str(intervalos[id_switch])]
['maximop']
             switch[id_switch]['desvio_minimob']=switch[id_switch][str(intervalos[id_switch])]
['minimob']
            switch[id_switch]['desvio_maximob']=switch[id_switch][str(intervalos[id_switch])]
['maximob']

    os.system("echo AQUI 2 >> ~/testesaida")
    os.system("echo Switch:"+ str(id_switch) + ">> ~/testesaida")
    os.system("echo intervalo:" + str(intervalos[id_switch]) +" >> ~/testesaida")
    now = datetime.now()
     os.system("echo Hora:" + str(now.hour)+":"+str(now.minute)+":"+str(now.second)+" >>
~/testesaida")
        os.system("echo chamadas_analise:" + str(chamadas_analise[id_switch]) +" >>
~/testesaida")
       os.system("echo desvio_minimop:" + str(switch[id_switch]['desvio_minimop']) +" >>
~/testesaida")
      os.system("echo desvio_maximop:" + str(switch[id_switch]['desvio_maximop']) +" >>
~/testesaida")
       os.system("echo desvio_minimob:" + str(switch[id_switch]['desvio_minimob']) +" >>
~/testesaida")
```

```
        os.system("echo desvio_maximob:" + str(switch[id_switch]['desvio_maximob']) +" >> ~/testesaida")
    os.system("echo . >> ~/testesaida")

  log.info(" Totais: %s bytes - %s pacotes - %s fluxos - chamada:%s - Aceita:%s - SwitchP: %s - SwitchB:%s", total_bytes, total_pacotes, web_flows, x, aceita_fluxo, total_diferenca_pacotes, total_diferenca_bytes)
  log.info("##########################################")

  #se encontrar anomalia no switch emite um aviso total_diferenca_pacotes
  try:
                if ((total_diferenca_pacotes < switch[id_switch]['desvio_minimop']) or (total_diferenca_pacotes > switch[id_switch]['desvio_maximop']) or (total_diferenca_bytes < switch[id_switch]['desvio_minimob']) or (total_diferenca_bytes > switch[id_switch]['desvio_maximob'])) and (switch_desvios>lim_desvios_switch):
    log.info("IDENTIFICANDO POSSIVEL ANOMALIA NO SWITCH! %s: Minp:%s e Maxp: %s | Minb:%s e Maxb:%s | P:%s | B:%s", id_switch, switch_pacote_min, switch_pacote_max, switch_byte_min, switch_byte_max,total_diferenca_pacotes,total_diferenca_bytes)

    os.system("echo intervalo:" + str(intervalos[id_switch]) +" >> ~/falso_positivo")
    now = datetime.now()
     os.system("echo Hora:" + str(now.hour)+":"+str(now.minute)+":"+str(now.second)+" >> ~/falso_positivo")
    os.system("echo Switch:"+ str(id_switch) + ">> ~/falso_positivo")
    os.system("echo MinP: " + str(switch_pacote_min) + " >> ~/falso_positivo")
    os.system("echo MaxP: " + str(switch_pacote_max) + " >> ~/falso_positivo")
    os.system("echo MinB: " + str(switch_byte_min) + " >> ~/falso_positivo")
    os.system("echo MaxB: " + str(switch_byte_max) + " >> ~/falso_positivo")
    os.system("echo Pacotes: " + str(total_diferenca_pacotes) + " >> ~/falso_positivo")
    os.system("echo Bytes: " + str(total_diferenca_bytes) + " >> ~/falso_positivo")
    os.system("echo Aceita Fluxo: " + str(aceita_fluxo) + " >> ~/falso_positivo")
    os.system("echo ---------------------------- >> ~/falso_positivo")
```

```
        log.info("Desvio atual - Pacotes: %s - %s | Bytes: %s - %s",switch[id_switch]
['desvio_minimop'],switch[id_switch]['desvio_maximop'],switch[id_switch]
['desvio_minimob'],switch[id_switch]['desvio_maximob'])
  except:
    log.info("")

  #armazena todos os valores coletados nas chamadas no arquivo, para criar o grafico
  if (switch_desvios>lim_desvios_switch):
    now = datetime.now()
                os.system("echo "+str(id_switch)+" Desvio:"+str(intervalos[id_switch])+"
Pacotes:"+str(total_diferenca_pacotes)+" "+str(now.hour)+":"+str(now.minute)
+":"+str(now.second)+" >> ~/analise_pacotes")
                os.system("echo "+str(id_switch)+" Desvio:"+str(intervalos[id_switch])+"
Bytes:"+str(total_diferenca_bytes)+" "+str(now.hour)+":"+str(now.minute)
+":"+str(now.second)+" >> ~/analise_bytes")

  #zera os valores para pegar a soma dos fluxos dentro do for
  total_diferenca_pacotes=0
  total_diferenca_bytes=0

  #permite novamente contar mais uma chamada para o switch
  conta_chamada=True

  #volta a nao aceitar bloqueio de desconhecido
  #aceita_bloqueio_conhecido=False

  #volta a ser True para bloquear o primeiro, se for ip desconhecido
  aceita_bloqueio=True

  if (portas_bloqueadas) and (atualiza_tabela==True):
    event.connection.send( of.ofp_flow_mod(command=of.OFPFC_DELETE))
    for i in portas_bloqueadas:
        event.connection.send( of.ofp_flow_mod(match=of.ofp_match(in_port=portas_bloque
adas[i])))
```

```
    log.info("PORTAS QUE DEVEM SER BLOQUEADOS: %s ", str(portas_bloqueadas))
    atualiza_tabela=False

    #vai diminuindo o tempo_bloqueio ate zero para limpar a tabela de fluxo e liberar novamente a porta que foi detectado o ataque
  if (tempo_bloqueio>0):
    tempo_bloqueio=tempo_bloqueio-1
    log.info("DIMINUINDO TEMPO EM %s", tempo_bloqueio)
  elif (tempo_bloqueio==0):
    event.connection.send(of.ofp_flow_mod(command=of.OFPFC_DELETE))
    os.system(" > ~/vitimas_conhecidas")
    log.info("DESBLOQUEANDO PORTA")
    portas_bloqueadas.clear()
    tempo_bloqueio=tempo_bloqueio-1

# handler to display port statistics received in JSON format
def _handle_portstats_received (event):
  stats = flow_stats_to_list(event.stats)
  log.debug("PortStatsReceived from %s: %s",
    dpidToStr(event.connection.dpid), stats)

# main functiont to launch the module

def launch ():
  from pox.lib.recoco import Timer

  #apaga os ips escaneados e faz um novo scan de ips
  palavra="> ~/ips_scan"
  os.system(palavra)
  palavra="echo '#' > ~/ips_scan_final"
  os.system(palavra)

  # attach handsers to listners
```

```
core.openflow.addListenerByName("FlowStatsReceived",
  _handle_flowstats_received)
core.openflow.addListenerByName("PortStatsReceived",
  _handle_portstats_received)

palavra="echo 0 > ~/chamadas"
os.system(palavra)

os.system("> ~/desvios")
os.system("> ~/testesaida")
os.system("> ~/falso_positivo")
os.system("> ~/analise_pacotes")
os.system("> ~/analise_bytes")
os.system("> ~/atacantes_conhecidos")
os.system("> ~/atacantes_desconhecidos")
os.system(" > ~/vitimas_conhecidas")

# timer set to execute every five seconds
Timer(leitura, _timer_func, recurring=True)
```