UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

ROBERTO OPPERMANN CORDONI

Detecção de Ataques DDoS Baseados em Amplificação NTP Inteiramente no Plano de Dados

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof. Dr. Luciano Paschoal Gaspary

Porto Alegre
2018

“O sucesso não é definitivo, o fracasso não é fatal: é a coragem para continuar que conta”

— SIR WINSTON CHURCHILL

AGRADECIMENTOS

Aos meus pais, pelo incansável esforço para que nunca me faltasse nada e eu chegasse até aqui.

As minhas irmãs, por se fazerem disponíveis sempre que eu precisei delas.

Ao meu orientador, Luciano Gaspary, pela paciência e orientação ao longo do desenvolvimento deste trabalho.

E por fim, como estudante de uma universidade pública, a todos os brasileiros e brasileiras que financiaram minha graduação.

RESUMO

Os ataques de negação de serviço distribuídos representam uma grande parcela dentre o total de ataques cibernéticos reportados nos últimos anos. Um tipo de ataque em especial tem chamado a atenção dos administradores de rede por ser substancialmente danoso: os ataques DDoS baseados em amplificação. Ao lançar um ataque desse tipo, atacantes conseguem atingir taxas de dados muito altas, tirando proveito de protocolos como NTP, DNS e SSDP. Atualmente, provedores de serviço utilizam técnicas baseadas em NetFlow e sFlow para detectar tais ameaças. Contudo, tais táticas possuem duas limitações principais: alto custo computacional para processamento em software dos pacotes que transitam pelos dispositivos de encaminhamento; e elevado tempo de reação a um ataque, devido ao distanciamento entre os sensores e os coletores de dados. Buscando eliminar as barreiras impostas pelo processamento feito fora dos dispositivos, uma nova geração de equipamentos de encaminhamento tem sido desenvolvida. Esses equipamentos conseguem trabalhar com altas taxas de dados e podem ser programados através de uma linguagem, denominada P4, que permite especificar como pacotes devem ser analisados e processados. Neste trabalho, propõe-se dois mecanismos, implementados em P4, para detecção de ataques de negação de serviço baseados em amplificação que tiram proveito do protocolo NTP. O primeiro desses mecanismos atua nas proximidades do servidor usado como amplificador, enquanto o segundo encontra-se localizado nas imediações da vítima do ataque. A acurácia e a eficiência desses mecanismos foram avaliadas através de três métricas: falsos negativos, falsos positivos e tempo de detecção. De maneira geral, ambos mecanismos atingem melhores resultados quando o ataque possui uma "assinatura" muito agressiva e, através de experimentos realizados, foi identificado que quanto mais intenso o ataque, menor o número de falsos positivos observados. Por fim, foi apresentada uma comparação demonstrando que o tempo de detecção em P4 é menor do que o tempo necessário para uma possível implementação desse mecanismo em OpenFlow convencional.

Palavras-chave: Redes de computadores. Software-Defined Networking. P4. DDoS.

ABSTRACT

Distributed denial of service attacks account for a large share of the total number of cyber attacks reported in recent years. One particular type of attack has caught the attention of network administrators because of its devastating effect: DDoS attacks based on amplification. By launching such an attack, attackers can achieve very high data rates, taking advantage of protocols such as NTP, DNS and SSDP. Nowadays, service providers make use of NetFlow and sFlow-based techniques to detect such threats. However, these techniques present two main limitations: high computational cost for packets processing; and high response time due to the distance between data sensors and data collectors. Seeking to eliminate those barriers, a new generation of routing equipments has been developed. These devices are able to operate at high data rates and can be programmed through P4, a programming language that allows specifying how packets may be analyzed and processed. In this work, it is proposed two mechanisms, implemented in P4, to detect DDoS amplification attacks that take advantage of the NTP protocol. The first of these mechanisms acts near the server used as amplifier, while the second one is located nearby the victim of the attack. The accuracy and efficiency of these mechanisms were evaluated through three metrics: false negatives, false positives and detection time. In general, they all achieve better results when the attack has a very aggressive "signature". Through experiments, it was identified that the more intense the attack, the lower is the number of false positives observed. Finally, a comparison was made showing that the detection time in P4 is inferior to the time required for a possible implementation of this mechanism in conventional OpenFlow.

Keywords: Computer networks. Software-Defined Networking. P4. DDoS.

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
CPU	Central Processing Unit
DDoS	Distributed Denial of Service
DNS	Domain Name System
DRDoS	Distributed Reflective Denial of Service
FPGA	Field Programmable Gate Array
Gbps	Gigabits per Second
HTTP	Hypertext Tranfer Protocol
IP	Internet Protocol
Kbps	Kilobits per Second
NPU	Network Processing Unit
NTP	Network Time Protocol
P4	Programming Protocol-independent Packet Processors
SDN	Software-Defined Networking
SSDP	Simple Service Discovery Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

LISTA DE ALGORITMOS

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Aliada com a rápida expansão da internet ocorrida nas últimas décadas, a quantidade de ataques cibernéticos reportados também apresentou um alto crescimento. O prejuízo associado a tais ataques não fica limitado à esfera financeira, como a perda de milhares de dólares caso um serviço fique fora do ar, mas também pode causar despesas no sentido da perda de clientes/reputação de uma certa empresa. Desde o início de 2015, o número total de ataques de negação de serviço identificados vem aumentando com uma média de quase 10% por trimestre, alcançando um total de mais de 4.000 ataques relatados no segundo trimestre de 2017 (AKAMAI TECHNOLOGIES INC., 2017b).

O principal objetivo de ataques de negação de serviço, em especial os distribuídos (DDoS), é impedir o acesso de usuários legítimos a algum recurso de rede. Dentre as diversas subcategorias de ataques DDoS, existem os ataques de negação de serviço reflexivos (DRDoS). Nesse tipo de ataque, atacantes enviam requisições para servidores de acesso público, geralmente operando com protocolos baseados em UDP, forjando seu endereço IP de origem com o endereço da vítima. O servidor, por sua vez, acaba por despropositadamente inundar a rede da vítima com respostas válidas, sem que ela as tenha requisitado. Um caso particular de ataques de reflexão, bastante empregado atualmente, é o que explora o conceito de amplificação. Em tais ataques, um amplificador envia respostas significativamente maiores que as requisições, fazendo com que seus efeitos sejam especialmente danosos. Segundo relatórios recentes, há uma clara tendência no aumento do número de ataques DRDoS. No início de 2017, esses ataques chegaram a representar 57% do número total de ataques de negação de serviço (AKAMAI TECHNOLOGIES INC., 2017a).

Atualmente, ferramentas baseadas em NetFlow e sFlow são utilizadas por mais de 85% dos provedores de serviço para a detecção de ataques/ameaças (ARBOR NETW. INC., 2017). Esse tipo de ferramenta consiste em amostrar e armazenar um subconjunto dos pacotes que passam por dispositivos de encaminhamento e periodicamente enviar esses registros para um equipamento coletor. Contudo, o custo computacional elevado para o processamento em software de tais amostras torna inviável a amostragem de pacotes em altas taxas. Na prática, essas ferramentas amostram em média 1 a cada 100 pacotes, o que torna esse método menos acurado (SIVARAMAN et al., 2017). Mais importante, esse distanciamento entre sensor e coletor reduz a capacidade de reagir a um ataque de maneira mais rápida, já que o coletor necessita requisitar informações, receber respostas e

só então avaliar os resultados. Vale ressaltar, ainda, que o referido distanciamento insere uma sobrecarga não desprezível na rede devido às mensagens trocadas entre dispositivos de encaminhamento e software coletor.

Procurando eliminar as barreiras impostas pelo processamento feito fora dos dispositivos, a indústria vem apresentando recentemente uma nova geração de equipamentos de encaminhamento que podem operar a taxas de 10-100 Gbps por porta (BAREFOOT NETWORKS, 2017). Tais dispositivos podem ser programados para executar diversos comandos sobre cada pacote, por meio de uma linguagem denominada P4 (BOSSHART et al., 2014). P4 permite especificar como pacotes devem ser analisados e processados, via uma sequência de *parsers* e tabelas do tipo *match+action*.

Apesar de seus potenciais benefícios, pouco ainda tem sido explorado na direção do uso de planos de dados programáveis como "plataformas" propícias para detecção de intrusão. Em teoria, a análise de pacotes em altas taxas e a possibilidade de programação do dispositivo para manter diferentes informações sobre o tráfego traria agilidade e rapidez no processo de detecção de ataques tais como os de negação de serviço.

Neste trabalho, objetiva-se investigar o potencial do emprego de planos de dados programáveis P4 para realizar detecção de intrusão. Mais especificamente, realiza-se a prototipação de "programas" de detecção de ataques DDoS baseados em amplificação e analisa-se a acurácia e a eficiência dos mecanismos implementados. Para apoiar a obtenção de resultados, o presente trabalho contribui, ainda, com o projeto e o desenvolvimento de um ambiente de experimentação voltado à avaliação simplificada de diferentes cenários.

O restante do trabalho está organizado como segue. O Capítulo 2 apresenta os conceitos fundamentais para a compreensão do trabalho, assim como uma síntese do estado da arte. O Capítulo 3 propõe dois mecanismos de detecção. Um para atuar próximo ao serviço comprometido, e outro para atuar nas imediações da vítima. Já o Capítulo 4 apresenta a implementação, em P4, das soluções desenvolvidas. O Capítulo 5 traz a metodologia de avaliação, bem como os resultados obtidos. Por fim, o Capítulo 6 demonstra uma análise geral do desenvolvimento do trabalho, incluindo a exemplificação de trabalhos futuros.

2 FUNDAMENTOS

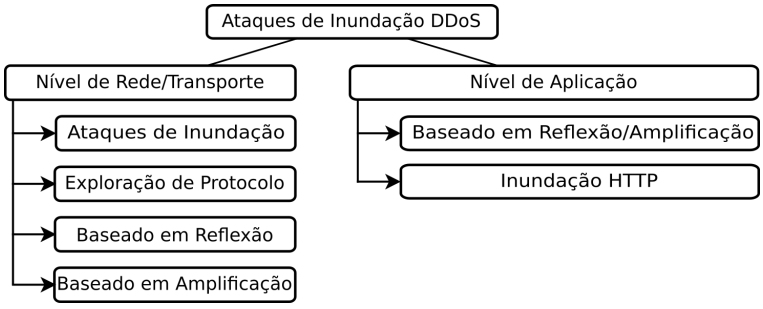
Este capítulo apresenta os conceitos fundamentais relacionados ao tema deste trabalho. Inicialmente expõe-se a arquitetura dos ataques de negação de serviço. Em seguida, identifica-se características referentes aos planos de dados programáveis e à linguagem P4. Por fim, demonstra-se o estado da arte no que diz respeito à detecção de intrusão utilizando P4.

2.1 Ataques de Negação de Serviço

Em ataques de negação de serviço, um atacante com motivação financeira, política ou puramente destrutiva visa a degradar o acesso a um serviço através de ações maliciosas, comprometendo a disponibilidade de algum recurso da vítima. Segundo Douglieris e Mitrokotsa (2004), ataques de negação de serviço podem partir de uma ou mais fontes. Em ataques individuais (*Denial of Service* - DoS), um único computador é utilizado para executar ações maliciosas contra uma ou mais vítimas. Já os distribuídos (*Distributed Denial of Service* - DDoS) se diferenciam do último por utilizarem muitos dispositivos computacionais para iniciar, coordenadamente, um ataque. Essa última forma se tornou popular por ser mais efetiva que DoS e também por tornar sua detecção/mitigação mais complexa. Usualmente, esse ataque coordenado é executado através de um conjunto de máquinas infectadas, que permitem o controle remoto de seus recursos computacionais pelo atacante, conhecido como *botnet*.

A Figura 2.1 ilustra uma classificação aprofundada para ataques DDoS, proposta por Zargar, Joshi e Tipper (2013). Essa taxonomia serve de ponto de partida para explicar alguns conceitos importantes no contexto deste trabalho. Inicialmente, ataques de nega-

Figura 2.1: Classificaçao de ataques DDoS.



Fonte: Adaptado pelo autor.

ção de serviço distribuídos podem ser classificados em duas categorias que se diferenciam pelo nível de atuação do protocolo utilizado. A primeira delas se refere a ataques em nível de rede/transporte. Os ataques nessa categoria visam a interrupção da conectividade de um usuário legítimo via exaustão de largura de banda ou algum outro recurso de rede. A lista de ataques nesse nível inclui: ataques de inundação (esgotamento de banda da vítima), exploração de protocolo (*bugs* em certos protocolos), ataques baseados em reflexão (IP *spoofing*) e, por fim, ataques baseados em amplificação (amplificação de tráfego).

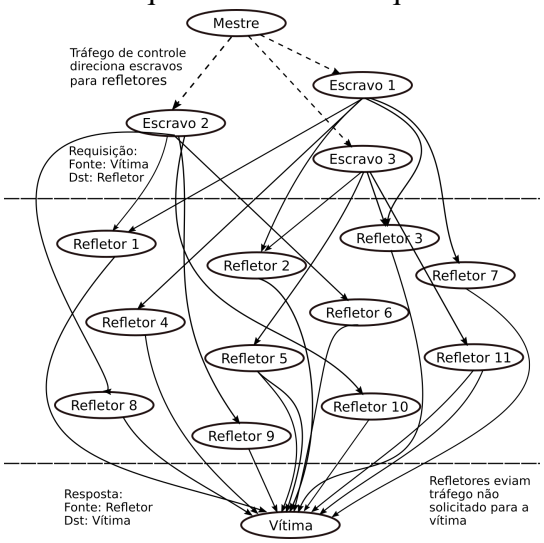
Já em ataques em nível de aplicação, atacantes exploram características específicas de determinados protocolos para saturar os recursos de um servidor (sockets, CPU, memória), causando a degradação do serviço oferecido pelo mesmo (ZARGAR; JOSHI; TIPPER, 2013). Estão inclusos nessa categoria: ataques baseados em amplificação/reflexão (IP *spoofing* e amplificação) e também ataques de inundação HTTP (exploração de diferentes características do protocolo).

Ataques baseados em reflexão. Este trabalho aborda mais especificamente a subclasse de ataques de aplicação baseados em reflexão/amplificação. Segundo Rossow (2014), em ataques de natureza reflexiva (*Distributed Reflective Denial of Service* – DRDoS), o atacante não envia tráfego diretamente para a vítima, mas sim para diversos servidores de acesso público baseados em UDP. Ele se aproveita do fato de que esses servidores respondem as requisições sem validar a identidade do cliente. Com isso, forja o endereço IP origem de suas mensagens com o endereço da vítima. Assim, o servidor (também chamado de refletor) inadvertidamente direciona o tráfego de resposta para o alvo do ataque (ROSSOW, 2014).

Além do grau de indireção criado pela falsificação da origem, outro ponto que motiva o uso de refletores é a taxa de pacotes necessária pelos mesmos para efetuar o ataque. Segundo Paxson (2001), devido ao maior número de refletores em relação ao de escravos, a taxa utilizada por cada refletor seria comparativamente menor do que a taxa necessária por cada escravo caso o mesmo tentasse inundar a vítima diretamente.

A Figura 2.2 demonstra a arquitetura do ataque de reflexão. Como pode ser observado, diversos atacantes pertencentes a uma *botnet* enviam requisições em direção a diversos refletores. Como o endereço IP de origem dessas requisições está forjado, o refletor acaba por direcionar sua resposta para a vítima, e não ao atacante. Tal ação, executada por centenas ou até milhares de fontes, resulta em uma elevada quantidade de tráfego na rede de destino.

Figura 2.2: Arquitetura de um ataque de reflexão.



Adaptado de: Paxson (2001).

Ataques baseados em amplificação. Ataques baseados em reflexão podem ser ainda mais danosos na medida que exploram um segundo conceito: o de amplificação. De acordo com Rossow (2014), atacantes tiram proveito de servidores que não só refletem, mas que também amplificam o volume do tráfego em direção à vítima (chamados de amplificadores). Tipicamente, eles utilizam protocolos cujas requisições, de tamanho pequeno, são capazes de gerar respostas substancialmente maiores. Dessa forma, atacantes não despendem de muitos recursos para promover um ataque bem-sucedido contra a vítima. A razão entre o tamanho da resposta e a sua respectiva requisição é chamada de fator de amplificação. Tal métrica é uma indicação direta da efetividade do ataque. Quanto maior o fator de amplificação, maior o consumo dos recursos da vítima.

$$Fator\,de\,Amplificacão = \frac{Tamanho\,(resposta)}{Tamanho\,(requisicão)}$$

Rossow (2014) demonstrou que existem mais de 14 protocolos/serviços baseados em UDP suscetíveis à amplificação. Dentre eles, os mais efetivos são: DNS (*Domain Name System*), NTP (*Network Time Protocol*) e SSDP (*Simple Service Discovery Protocol*).

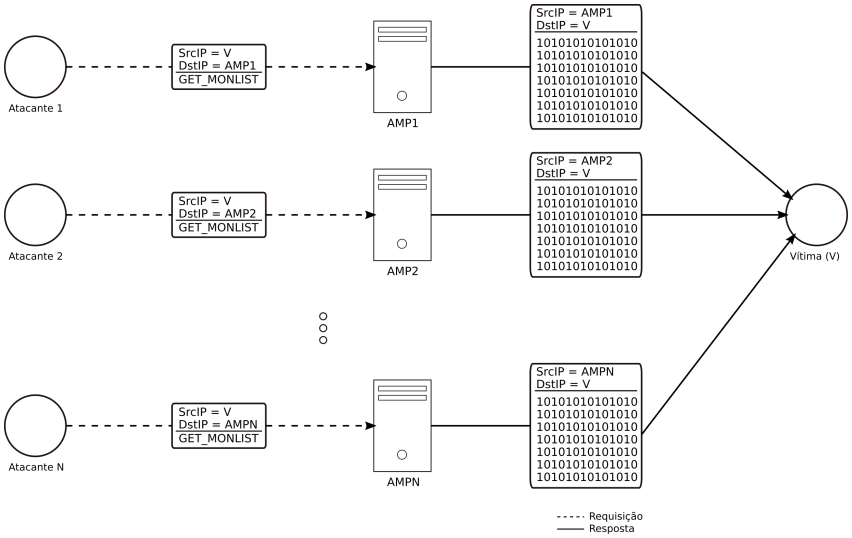
Ataque NTP Monlist. Feita a introdução sobre ataques de reflexão e amplificação, agora serão apresentados os conceitos relacionados aos ataques baseados no protocolo NTP. Esse é o protocolo padrão para serviços de sincronização de relógio, utilizado tanto pela indústria quanto por dispositivos de uso pessoal. Seu princípio básico de funciona-

mento é relativamente simples: clientes enviam requisições contendo o *timestamp* obtido com seu próprio relógio para servidores NTP, que respondem com seu respectivo *timestamp* e o instante de tempo no qual o pacote foi enviado de volta. Baseado nesses parâmetros, o cliente consegue calcular a diferença de tempo entre o relógio do servidor e o seu próprio relógio.

Além da funcionalidade acima, o protocolo implementa uma série de comandos usados para fins de monitoração. Dentre eles, o mais relevante para esse trabalho é o comando `monlist` (MON_GETLIST), que encontra-se habilitado por padrão em grande parte dos servidores. O comando `monlist` opera no que se chama modo privado. Esse modo é usado para a troca de informações de monitoração entre cliente e servidor. Quando um servidor recebe esse comando, ele responde essa mensagem com estatísticas sobre os últimos clientes que se conectaram a ele, tais como endereço IP, versão do protocolo utilizada e número de requisições. Essa mensagem de resposta pode estar dividida em até 100 datagramas UDP de 440 *bytes* cada.

A relação entre o tamanho dos pacotes trocados entre cliente e servidor pode ser extremamente discrepante, visto que uma requisição contém apenas 8 *bytes*, enquanto uma resposta pode conter centenas de *bytes*, dependendo das informações armazenadas no servidor no momento da interação. Essas características fazem do NTP uma ferramenta ideal para ataques de negação de serviço baseados em reflexão/amplificação. Como um exemplo do potencial desses ataques, pode-se citar o ocorrido em fevereiro de 2014 contra a empresa francesa CloudFlare, em que foi observado um pico de 400 Gbps nos dispositivos da vítima (PRINCE, 2014).

Figura 2.3: Exemplo de ataque de amplificação.



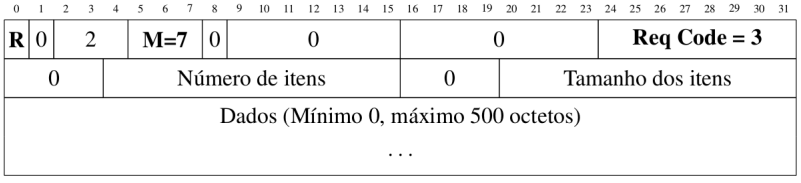
Fonte: Elaborado pelo autor.

A Figura 2.3 ilustra como o NTP é explorado para um ataque de amplificação. Como pode ser observado, o Atacante1 envia uma requisição NTP do tipo GET MONLIST para o amplifcador AMP1, falsificando o endereço IP de origem com o endereço da vítima V. O amplificador, por sua vez, envia para o suposto remetente da mensagem (V) um pacote de resposta cujo tamanho é ordens de grandeza maior que a respectiva requisição. Expandindo essa situação para milhares de atacantes, o resultado é uma inundação de pacotes em V. De acordo com experimentos realizados por Rossow (2014), requisições `monlist` conseguem amplificar o tráfego por fatores de até 4.670.

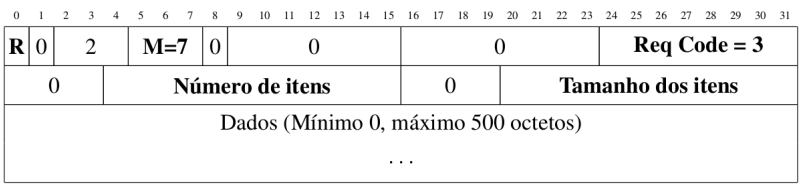
Aprofundando a compreensão desse ataque, a Figura 2.4 ilustra o formato de uma requisição e de uma resposta no contexto de um ataque NTP `monlist`. Observa-se que na mensagem de requisição da Figura 2.4a, o atacante essencialmente envia uma mensagem "normal" solicitando pela lista de monitoração. Os campos importantes para essa mensagem estão ressaltados na figura. São eles: R = 0, indicando que se trata de uma requisição; Mode = 7 identificando uma operação no modo privado; e, por fim, o campo Req Code = 3 indica o código da requisição `monlist`. Os campos Número de itens e Tamanho dos itens são ambos zerados para solicitações.

A mensagem de resposta, exemplificada na Figura 2.4b contém algumas diferenças com relação à respectiva requisição. A primeira delas se refere ao campo R = 1, que indica uma resposta. Além disso, tem-se dois campos de cabeçalho que indicam a quantidade de informação contida no campo de dados. O primeiro deles, *Número de items*, determina a quantidade de entradas na lista de monitoração. Já o campo *Tamanho dos itens* informa o tamanho, em *bytes*, de cada entrada dessa lista. Dessa forma, pode-se calcular o tamanho do campo de dados multiplicando esses dois últimos valores.

Figura 2.4: Exemplo de mensagens NTP MONLIST



(a) Requisição NTP MONLIST



(b) Resposta NTP MONLIST

Fonte: Elaborado pelo autor.

Após apresentadas as principais definições para ataques de negação de serviço, apresenta-se a seguir um novo paradigma de processamento nos dispositivos de encaminhamento, conhecido como plano de dados programável.

2.2 Programação do Plano de Dados com P4

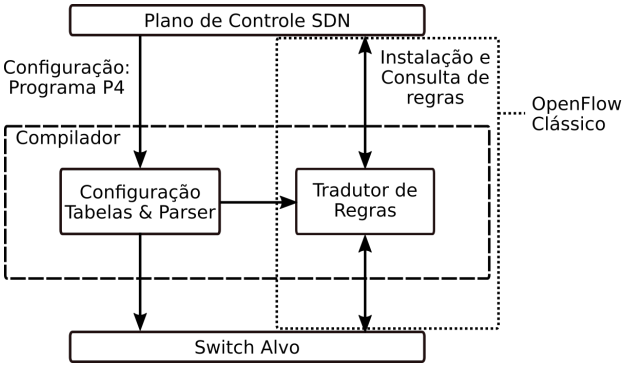
A programabilidade de redes é um assunto que vem sendo abordado há mais de vinte anos pela comunidade de pesquisa. Mais recentemente, o paradigma de redes definidas por software (SDN - *Software Defined Networking*) têm causado uma revolução na maneira com que se projetam e se gerenciam redes. Como uma das principais características dessa arquitetura, pode-se citar o desacoplamento entre o plano de controle, que decide como manipular o tráfego, e o plano de dados, cuja função é o encaminhamento de tráfego baseado nas decisões do plano de controle (FEAMSTER; REXFORD; ZEGURA, 2014).

Em redes SDN, o plano de controle pode ser visto como um software que gerencia múltiplos dispositivos do plano de dados (roteadores, *switches*, *etc*), utilizando APIs definidas para esse propósito, como o OpenFlow. Um dispositivo com suporte a OpenFlow pode se comportar como um roteador, *switch*, *firewall*, *etc* dependendo das regras que lhe foram instaladas pelo controlador de rede. Em sua primeira especificação, o protocolo OpenFlow contava com pouquíssimas tabelas e campos de cabeçalho. Porém, com o intuito de aperfeiçoar a comunicação entre os dispositivos de rede e o controlador, o protocolo foi sendo aprimorado e muitos outros tipos de cabeçalhos foram sendo adicionados à especificação, totalizando mais de 40 tipos diferentes até o final de 2013.

Bosshart et al. (2014) declararam que em vez de continuamente aumentar a especificação do OpenFlow, os dispositivos de rede deveriam suportar mecanismos mais flexíveis de análise de pacotes. O plano de controle definiria tal analisador utilizando uma nova interface de comunicação com o dispositivo. Essa interface deveria ser simples, elegante e mais difícil de se tornar obsoleta do que a atual especificação do OpenFlow. Dessa forma, os autores propuseram uma linguagem de alto nível para programar processadores de pacotes independentes de protocolo, denominada P4 (do inglês, *Programming Protocol-independent Packet Processors*).

Um dos principais requisitos para aplicações ligadas ao plano de dados, onde P4 foi planejado para atuar, é a capacidade de processar pacotes a uma taxa extremamente alta (tipicamente entre 10 e 100 Gbps, em dezenas de portas simultaneamente). Para

Figura 2.5: Linguagem P4 para configuração de *switches*.



Adaptado de: Bosshart et al. (2014).

atingir essas taxas, algoritmos de encaminhamento de pacotes eram usualmente implementados usando hardware dedicado. Contudo, o projeto de hardware é uma atividade complexa e de difícil reconfiguração em campo. Para testar e lançar um novo protocolo, ou até mesmo para modificar um sistema existente, pesquisadores eram obrigados a se dedicar a novos projetos de hardware (uma atividade demorada e onerosa). Para reduzir o tempo entre potenciais ciclos de inovações, a comunidade científica tentou durante muitos anos construir roteadores programáveis para que fosse possível o desenvolvimento de novas aplicações para dispositivos já finalizados.

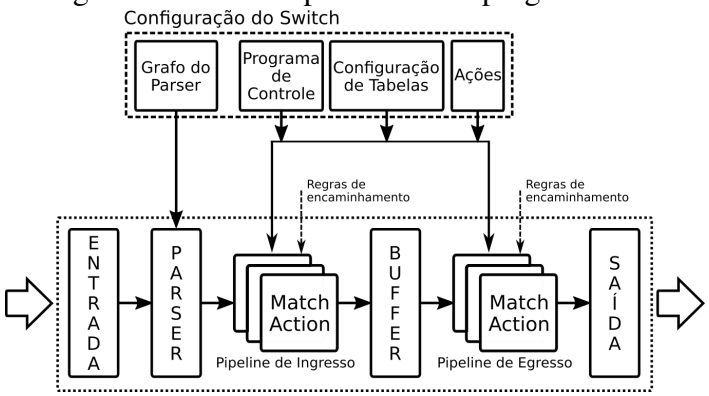
De acordo com Sivaraman et al. (2017), diversas pesquisas na área de roteadores programáveis resultaram em equipamentos cujo desempenho era diversas ordens de grandeza abaixo do necessário, e por isso nunca foram realmente utilizados em produção. Todavia, recentemente novos tipos de *chips* ASICs (do inglês, *Application Specific Integrated Circuits*) programáveis foram sendo disponibilizados pela indústria. Esses novos dispositivos são capazes de competir, em termos de desempenho, com equipamentos de funções fixas (*fixed-functions*) distribuídos atualmente. Em termos de programabilidade, esses novos *chips* permitem a um operador de rede especificar como analisar/encaminhar pacotes sem restrições em relação ao formato de protocolo ou ao conjunto de ações que podem ser executadas quando há correspondência de pacotes em uma tabela do tipo *match-action*.

A linguagem P4 pode então usar esse novo paradigma para configurar um dispositivo de encaminhamento, determinando de maneira flexível como pacotes devem ser processados e ao mesmo tempo garantindo alta taxa de dados. Assim, o programador não fica preso a um certo conjunto fixado de instruções oferecido por um equipamento não programável. De maneira geral, a Figura 2.5 demonstra a relação entre P4 e APIs existentes (como OpenFlow) utilizadas para popular as tabelas de encaminhamento do

equipamento programável. Os três principais objetivos da linguagem são:

- **Reconfigurabilidade**: O controlador deve possuir a capacidade de configurar e modificar as capacidades do dispositivo.
- **Independência de protocolo**: O equipamento não deve ser limitado em termos de protocolos suportados. Em vez disso, o controlador deve ser capaz de especificar: o *parser* para os possíveis tipos de formatos de cabeçalho e também ações a serem executadas sobre esses cabeçalhos.
- **Independência de hardware**: Não deve ser necessário para o controlador ter conhecimento do hardware do dispositivo a ser configurado. Isso deve ser trabalho do compilador que traduz código P4 em linguagem de máquina.

Figura 2.6: Modelo para *switches* programáveis.



Adaptado de: Bosshart et al. (2014).

P4: Modelo para Processadores de Pacotes. O modelo abstrato construído pelos idealizadores do P4 define, de uma forma genérica, como pacotes devem ser processados em diferentes dispositivos de encaminhamento (roteadores, *switches*) e tecnologias (*fixed-functions* ASICs, *Network Processing Units* - NPUs, Field Programmable Gata Arrays - FPGAs). Conforme exemplificado na Figura 2.6, esse modelo encaminha dados por meio de um analisador de pacotes programável, composto por múltiplos estágios de tabelas do tipo *match-action* (organizadas em série, paralelo ou uma combinação de ambos). De acordo com essa figura, pacotes ingressantes são primeiramente direcionados para o analisador (*parser*). Ele, então, extrai os cabeçalhos do pacote e os envia para as tabelas de *match-action*. Pode-se dizer que por definir os tipos de campos a serem extraídos, o *parser* caracteriza o programa e determina os tipos de protocolos que o dispositivo suporta.

As tabelas de *match-action* são divididas em *ingress* e *egress*. As tabelas de ingresso, além de determinarem a porta de saída e a fila de destino dos pacotes, também

estabelecem ações a serem tomadas para os mesmos. Algumas alternativas possíveis são: encaminhamento, replicação, rejeição, *etc*. Já as tabelas de egresso podem ser utilizadas para modificar cabeçalhos de pacotes, se necessário. Além disso, é possível ainda utilizar estruturas para manter informações sobre determinado pacote/fluxo, como registradores e contadores.

Além das funcionalidades já mencionadas, a linguagem possibilita que pacotes carreguem consigo informações adicionais por entre os estágios do *pipeline*. Esse tipo de informação é chamada de metadado (*metadata*). Esses metadados contêm informações essenciais sobre o pacote ingressante, tais como a porta de entrada, a porta de saída, o *timestamp* ou qualquer outro tipo de dado importante para o processamento do mesmo.

Componentes da Arquitetura P4. Depois de explicado o modelo abstrato idealizado para P4, o texto a seguir busca definir os aspectos necessários para a construção de um programa que esteja de acordo com esse modelo. Segundo Bosshart et al. (2014), um programa possui cinco componentes básicos, como segue:

- **Cabeçalhos (*headers*)**: a construção de um programa começa com a especificação dos formatos de cabeçalhos válidos. Para cada cabeçalho deve-se definir uma lista ordenada contendo os diferentes campos acompanhados de seus respectivos tamanhos.
- **Analisador (*parser*)**: P4 implementa a análise de cabeçalhos como uma máquina de estados. Para isso, ele assume que o dispositivo de destino é capaz de implementar uma máquina capaz de analisar o cabeçalho de cada pacote do início ao fim, extraindo seus diversos campos um por um. O processo de análise do cabeçalho começa no estado inicial (*start*) e prossegue até que o estado final (*stop*) seja atingido, ou até que uma transição não definida seja encontrada (caracterizando um erro).
- **Tabelas**: as tabelas declaram como os valores obtidos previamente dos cabeçalhos devem ser usados e também quais ações devem ser tomadas caso haja correspondência em alguma tabela.
- **Ações**: a própria linguagem P4 já estabelece uma série de operações primitivas. Cada programa, por sua vez, pode agrupar um conjunto dessas primitivas e definir uma ação. Alguns exemplos de primitivas são: `remove_header`, que retira certo campo do cabeçalho de um pacote e, `increment`, que incrementa o valor de um campo.

- **Programa de controle**: depois de definidos o *parser*, as tabelas e as ações, o programa de controle é responsável por especificar o fluxo a ser seguido pelo pacote entre uma tabela e outra.

Os componentes supracitados, juntos, definem um programa escrito na linguagem P4. O compilador, então, é responsável por analisar esse programa e gerar a configuração final para o dispositivo de destino.

Em resumo, P4 mostra-se como uma alternativa para os *switches* capazes de reconhecer apenas um número limitado de cabeçalhos e que processam pacotes utilizando um conjunto predeterminado de ações. A linguagem propõe um avanço para o modo como dispositivos pertencentes ao plano de dados são configurados (e modificados) em campo. Os idealizadores da linguagem acreditam que, através de dispositivos P4, será possível expandir o potencial de redes definidas por software, assim como aumentar a sua flexibilidade.

2.3 Trabalhos Relacionados e Estado da Arte

Atualmente, diferentes autores apresentam trabalhos na área de detecção de ataques de negação de serviço. Nesta seção, realiza-se um levantamento dos trabalhos existentes, buscando identificar o que há de mais atual na área. Primeiramente, relaciona-se pesquisas que discutem e sumarizam diferentes tipos de ataques e seus respectivos mecanismos de defesa. Na sequência, avalia-se trabalhos precursores na área de detecção e mitigação de ataques DDoS que não utilizam conceitos de redes definidas por software. Para finalizar, analisa-se diversos estudos que utilizam técnicas baseadas em SDN/OpenFlow e P4.

Classificação de ataques. Os trabalhos de Zargar, Joshi e Tipper (2013) e Yan et al. (2016) buscam resumir e apresentar algumas classificações para os diferentes tipos de ataques de negação de serviço existentes, além de categorizar diferentes contramedidas para prevenir, detectar e mitigar os efeitos desses ataques. Do mesmo modo, porém com mais foco em ataques de amplificação, Rossow (2014) revisita protocolos baseados em UDP e revela 14 deles que podem ser empregados em ataques de DRDoS e amplificação. O autor também relata que existe um número enorme de dispositivos que podem ser usados como potenciais amplificadores para esses ataques. Em outra pesquisa, Kührer

et al. (2014) afirmam que os principais responsáveis por ataques de amplificação são as redes que não aplicam mecanismos de *egress filtering*. Em seus resultados, os autores confirmam a existência de 2.692 sistemas autônomos que permitem IP *spoofing*.

Detecção de ataques DDoS em redes tradicionais. Kreibich et al. (2005) introduzem a noção de assimetria de pacotes para detecção de tráfego não solicitado. Os autores propõe o cálculo de um indicador, baseado na medição do número de pacotes recebidos/enviados, para indicar um possível tráfego maligno. De foma análoga, Kambourakis et al. (2007) apresentam um mecanismo para detecção de ataques de amplificação que utiliza a correspondência entre requisições e respostas do protocolo DNS, de forma que respostas que não "casem" com nenhuma requisição são marcadas como suspeitas.

Já Rossow (2014), ao invés de contabilizar mensagens, analisa a quantidade de *bytes* enviados entre cliente e servidor. O autor utiliza técnicas baseadas em NetFlow para agregar fluxos e reportar um ataque em caso de tráfego suspeito. Existe também a pesquisa de Backes et al. (2016), que avalia a viabilidade de se usar *Hop Count Filtering* (HCF) para mitigar ataques DRDoS. Os autores construíram um modelo estatístico, baseado em *active probing*, que permite ao refletor estimar o número de *hops* entre ele e o suposto cliente da requisição. Assim, o refletor usa essa estimativa como uma forma de detectar requisições que tenham sido enviadas com endereço IP forjado.

Detecção de ataques DDoS em redes definidas por software. O advento das redes definidas por software trouxe um novo conjunto de possibilidades que podem ser exploradas para defesa contra possíveis ataques. Por exemplo, o desacoplamento entre o plano de dados e o plano de controle facilita a inovação ao prover uma rede programável em que é possível a implementação e experimentação de novas ideias para aplicações de segurança. Mais ainda, o uso de um controlador centralizado que possui uma visão global da rede favorece a construção de artifícios para a análise e a monitoração de tráfego. Por esses motivos, os próximos parágrafos discutem a utilização de SDN para detecção de ataques.

Braga, Mota e Passito (2010) propõem um método para detecção de ataques DDoS que utiliza *Self Organizing Maps* (SOM) e redes neurais. Esse método é implementado sobre uma rede NOX, onde *switches* OpenFlow mantém estatísticas sobre todos os fluxos ativos. Eventualmente, o controlador obtém tais dados e os usa como entrada para a rede neural, que classifica a informação recebida como ataque ou não.

Yao, Bi e Xiao (2011) propõem um mecanismo chamado *Virtual source Address*

Validation Edge (VAVE) para detecção de endereços IP falsificados. Nesse mecanismo, dispositivos OpenFlow são usados para formar um "perímetro de segurança". Assim, sempre que um pacote vier de fora desse perímetro, e não houver correspondência com nenhuma entrada na tabela de fluxos dos dispositivos, o pacote é direcionado para o controlador.

Em outra frente, Zaalouk et al. (2014) apresentam OrchSec, uma arquitetura baseada em "orquestração" que utiliza SDN e monitoração para o desenvolvimento de aplicações de segurança. Nesse trabalho, os autores abordam o problema de ataques de amplificação que utilizam o protocolo DNS da seguinte maneira: o controlador da rede monitora a quantidade de respostas DNS, e, assim que essa quantidade excede um certo limite, essas mensagens são redirecionadas para o Orquestrador (uma aplicação hierarquicamente "acima" do controlador) durante um curto período de tempo. Baseado no cálculo da média do tamanho dos pacotes, e na entropia dos endereços de destinos dos mesmos, o Orquestrador é capaz de identificar a ocorrência de um ataque e mitigar seus efeitos através de *rate-limiting*.

Acrescenta-se os estudos de Li et al. (2016), que propõem um mecanismo de monitoração baseado em NetFlow, chamada FlowRadar. Esse mecanismo, implementado em P4, consegue armazenar contadores para todos os fluxos existentes com pouca utilização de memória, usando *encoded flowsets*. Apesar do armazenamento de informações ser feito no plano de dados, é o controlador da rede que executa a decodificação e a análise dessas informações, que são coletadas em intervalos de tempo de aproximadamente 10ms.

Alta latência de SDN e advento de P4. Grande parte dos trabalhos de segurança em redes definidas por software se caracteriza por demandar um elemento central, o controlador, que busca estatísticas nos dispositivos gerenciados e recebe como resposta dados para análise. A partir dessas informações, o controlador é capaz de tirar conclusões sobre o estado atual da rede e, eventualmente, executar alguma ação para mitigação. Essa troca de informações entre controlador e *switches* introduz custos em termos de latência de comunicação que podem dificultar a detecção de intrusões em tempo razoável. Ademais, essa troca de informação pode ser explorada por atacantes para inundar a comunicação entre as partes, causando o chamado ataque de saturação do plano de controle, como mencionado por Shin et al. (2013).

Em vista das limitações mencionadas, a literatura se encaminha para uma classe

de trabalhos que busca transferir uma certa inteligência para o plano de dados de forma que ele possa atuar em estratégias de detecção que diminua as informações enviadas para o processador central. Tal tem sido possível graças ao advento e consolidação do paradigma de programabilidade de planos de dados e do amadurecimento de tecnologias como P4. Essa classe de trabalhos também visa a incluir mecanismos que sejam mais flexíveis com relação aos campos de cabeçalho utilizados. A seguir, são apresentadas e discutidas propostas trabalhos que lançam mão dessas características para projetar mecanismos de detecção.

Para melhorar o desempenho de seu detector, Giotis et al. (2014) propõem uma arquitetura modular que desmembra a função de coleta de dados do plano de controle, empregando monitoração do tipo sFlow. Comparada com abordagens OpenFlow tradicionais, a arquitetura proposta aperfeiçoa a coleta de dados e reduz a comunicação entre *switches* e controladores, eliminando possíveis sobrecargas do plano de controle.

Liu et al. (2016) apresentam UnivMon (*Universal Monitoring*), um *framework* para monitoração de fluxos que utiliza avanços recentes na área de *universal streaming*. Os autores formulam um único *sketch* que prova alcançar boa generalidade e alta fidelidade diante de um vasto espectro de tarefas de monitoração. Para identificar os top-k fluxos mais pesados (*Heavy Hitters*), eles dividem o cálculo entre elementos do plano de controle e do plano de dados (P4). Dessa maneira, o plano de dados passa a ser uma parte essencial na identificação de tais fluxos.

Na mesma linha do trabalho recém apresentado, Sivaraman et al. (2017) sugerem um algoritmo que armazena estatísticas para os k fluxos mais pesados, delimitado pelas restrições de uso de memória de *switches* programáveis. O algoritmo, denominado HashPipe, atinge seu objetivo ao identificar menos de 5% de falsos negativos e 0.001% de falsos positivos, utilizando somente operações internas do dispositivo P4 para isso. Popescu, Antichi e Moore (2017) buscam estender o trabalho de Sivaraman et al. (2017) ao utilizar a programabilidade do plano de dados para transformar o *switch* em um dispositivo "ativo", e não mais passivo. No algoritmo proposto, o plano de dados entra em contato com o plano de controle quando a detecção de uma atividade maliciosa for efetivada.

Afek, Bremler-Barr e Shaffir (2017) utilizam a programabilidade do plano de dados para abordar o problema de IP *spoofing*. Dentre diversos métodos, os autores lidam com inundações de pacotes DNS forjados da seguinte forma: quando um *switch* recebe uma requisição DNS UDP proveniente de uma origem que ainda não foi autenticada, ele força o cliente a repetir essa requisição, porém utilizando TCP. Após o *hand-shake* ter

sido completado, o dispositivo instala uma regra para permitir que futuras requisições desse cliente possam ser realizadas via UDP. É importante ressaltar que não há envolvimento com o controlador durante a fase de autenticação e que, assim como Zaalouk et al. (2014), o algoritmo proposto manipula campos de cabeçalho não usuais para dispositivos de encaminhamento.

A partir das explicações dos trabalhos existentes na área, é possível verificar a utilização do plano de dados para a construção de novas metodologias de análise em diversos aspectos relacionados à segurança em redes de computadores. Além disso, apesar dessa área possuir uma base relativamente sólida, ela oferece oportunidades interessantes de pesquisas e desenvolvimentos. Este trabalho contribui nesse sentido ao explorar diferentes estruturas da linguagem P4 para detecção *in-band* de ataques de negação de serviço baseados em amplificação NTP. Além disso, dá passos importantes na direção de permitir melhor compreensão da relação de compromisso entre implementação de mecanismos no plano de dados e desempenho.

3 MECANISMOS PARA DETECÇÃO DE ATAQUES DDOS BASEADOS EM AMPLIFICAÇÃO

Este capítulo tem por objetivo o detalhamento de mecanismos de detecção de ataques de amplificação propostos para este trabalho. Primeiramente, são apresentados dois cenários nos quais esses mecanismos podem estar localizados. Posteriormente, os mecanismos propostos são aprofundados e exemplificados. Por fim, busca-se fazer uma reflexão sobre alguns aspectos importantes no projeto dos mesmos.

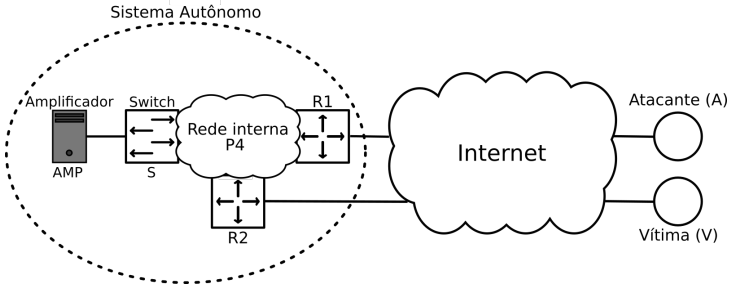
3.1 Cenários de Ataque

Mecanismos de detecção podem ser colocados em prática em diferentes lugares no caminho entre a vítima e a origem do ataque. Zargar, Joshi e Tipper (2013) classificaram essas diferentes localidades em: *(i)* rede de origem, *(ii)* rede de destino, *(iii)* meio da rede, e *(iv)* distribuída. As estratégias ditas na rede de origem visam empregar o mecanismo de detecção próximo da fonte do ataque para impedir o início de uma atividade maliciosa. Já as estratégias aplicadas na rede de destino buscam abordar o problema nas imediações da vítima. O sucesso da detecção nessa área é relativamente maior visto que a rede de destino concentra todo o tráfego malicioso destinado para ela. A terceira localidade citada pelos autores se refere a mecanismos aplicados dentro das redes de trânsito entre a origem e o destino do ataque. Por fim, as estratégias distribuídas são empregadas em múltiplas localidades ao mesmo tempo, buscando cooperar entre si para aumentar a eficácia e a eficiência da detecção (ZARGAR; JOSHI; TIPPER, 2013).

Neste trabalho, propõem-se duas estratégias de detecção a serem implantadas em localidades distintas. A primeira delas se localiza na rede de destino, enquanto a segunda se encontra posicionada nas proximidades do servidor NTP passível de amplificação. Não há uma classificação detalhada descrita na literatura para essa última abordagem. Contudo, por possuir características semelhantes a mecanismos estabelecidos na fonte do ataque, ela será categorizada neste trabalho como localizada na rede de origem. A seguir, apresenta-se os dois cenários descritos.

Cenário 1 - Detecção próxima ao serviço comprometido. A Figura 3.1 considera a visão de um sistema autônomo que possui múltiplos pontos de entrada e saída de tráfego

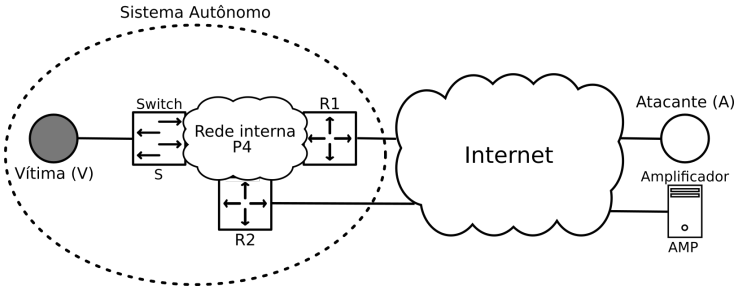
Figura 3.1: Cenário 1 - Detecção próxima ao serviço comprometido.



Fonte: Elaborado pelo autor.

para a internet, tais como os roteadores de borda identificados através de R1 e R2. Dentro desse cenário, ilustra-se uma rede interna ao sistema autônomo que possui todos os equipamentos, ou pelo menos um subconjunto deles, habilitados para P4. Conectado à essa sub-rede, encontra-se o serviço a ser usado por um atacante como amplificador, tal como o NTP, que deve possuir um endereço IP público de forma que seja acessível na internet. Dessa maneira, sugere-se que o mecanismo de detecção esteja localizado em um dos dispositivos P4 presentes na rede interna desse sistema autônomo. Além disso, é necessário que todo o tráfego proveniente ou direcionado para AMP seja agregado no equipamento no qual o mecanismo se encontra (S ou algum outro equipamento dentro dessa sub-rede). Essa exigência é necessária, pois, caso exista outra rota, o *switch* pode não possuir as informações de todas as mensagens trocadas por esse amplificador e, portanto, a acurácia da detecção pode ficar comprometida. Nesse cenário, atacantes e vítimas podem estar localizados em qualquer lugar da rede, pois tanto mensagens de requisição quanto resposta necessariamente transitam pelo dispositivo de encaminhamento S.

Figura 3.2: Cenário 2 - Detecção próxima à vítima.



Fonte: Elaborado pelo autor.

Cenário 2 - Detecção próxima à vítima. A Figura 3.2, por sua vez, exemplifica o cenário em que o mecanismo de detecção se encontra localizado próximo da vítima. Note que nesse contexto, a vítima (V) encontra-se conectada à sub-rede do sistema autônomo,

e o serviço amplificador, em qualquer lugar da topologia. Quase todas as propriedades descritas na perspectiva anterior também se aplicam a esse cenário. A única exceção fica por conta da localização do atacante. Agora, tem-se a restrição de que o mesmo não pode estar estabelecido na mesma sub-rede que a vítima. Como será explicado posteriormente, isso se deve ao fato de que caso essa situação ocorra, o dispositivo pode erroneamente considerar o tráfego malicioso como legítimo.

Após explicados os diferentes casos de análise, as próximas seções deste capítulo apresentam detalhadamente as duas estratégias propostas, baseadas no cenário em qual se inserem.

3.2 Detecção Próxima ao Serviço Comprometido

Depois de exemplificados os cenários em que os mecanismos propostos estão inseridos, detalha-se agora um algoritmo que detecta ataques de amplificação em casos como o descrito no Cenário 1. Ressalta-se que o algoritmo é executado toda vez que um pacote entra no *pipeline* de ingresso do *switch* S, após ter sido processado pelo *parser*. Sendo assim, os dados de entrada do algoritmo são os diferentes campos de cabeçalhos extraídos dos pacotes ingressantes.

Mecanismo proposto. Em resumo, o mecanismo proposto busca armazenar os *bytes* recebidos através de requisições e respostas `monlist` para uma determinada vítima. Em mensagens de resposta, além dos *bytes*, armazena-se também o momento (*timestamp*) em que a mesma ingressou no *switch*. Então, detecta-se um ataque caso: *(i)* o tempo decorrido entre duas respostas seja menor do que um limite aceitável, e *(ii)* a quantidade de *bytes* recebidos por respostas seja substancialmente maior que a quantidade recebida por requisições (também definida através de um valor limite). A fim de evitar que os contadores do mecanismo cresçam indefinidamente, caso o tempo decorrido entre duas respostas seja longo o suficiente, se reiniciam os contadores de *bytes* e *timestamps* associados.

Para melhor entendimento do processo de detecção, demonstra-se os passos executados com o auxílio do Algoritmo 1, que materializa o mecanismo proposto. De acordo com o mesmo, a primeira etapa consiste em verificar se o pacote ingressante se trata de uma mensagem NTP válida (linha 6 do Algoritmo 1). Em caso negativo, o algoritmo prontamente se encerra, evitando processamento desnecessário. Em caso positivo, o próximo passo é verificar se a mensagem se refere a uma requisição ou uma resposta (linhas

7 e 11). Tratando-se de uma requisição, soma-se o tamanho da mensagem (em *bytes*) ao registrador que armazena, para cada vítima, a quantidade de *bytes* recebida através de requisições (linha 9). Porém, caso a mensagem seja uma resposta, primeiramente deve-se atualizar o valor do registrador que armazena o *timestamp* da última resposta recebida (linhas 13 e 14).

Algoritmo 1: Algoritmo para o detector localizado próximo ao servidor NTP.

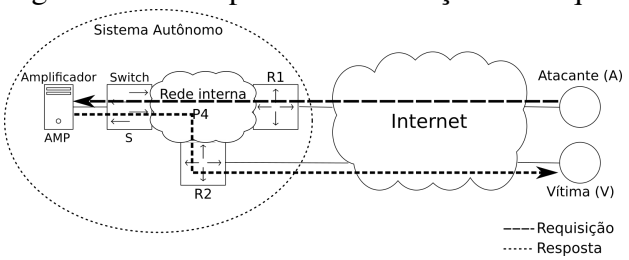
```
    entrada: header: refere-se ao cabeçalho do pacote que acabou de ingressar no
             pipeline do switch, após ter passado pelo parser.
 1  for indice = 0 to tamanho(requisições) do
 2      requisições [indice] = 0;
 3      respostas [indice] = 0;
 4      timestampResposta [indice] = 0;
 5  end
 6  if cabeçalho.NTP.Valido() then
 7      if cabeçalho.NTP.requestCode == REQUISICAO_MONLIST then
 8          indice = hash(cabeçalho.IPorigem);
 9          requisições [indice] = requisições [indice] +
             TamanhoRequisicao;
10      end
11      if cabeçalho.NTP.requestCode == RESPOSTA_MONLIST then
12          indice = hash(cabeçalho.IPdestino);
13          timestampAntigo = timestampResposta [indice];
14          timestampResposta [indice] = timestampAtual;
15          if timestampResposta [indice] − timestampAntigo <
             TS_THRESHOLD then
16              respostas [indice] = respostas [indice] +
                 TamanhoResposta;
17              if respostas [indice] − requisições [indice] >
                 BYTES_THRESHOLD then
18                  Ataque Detectado;
19                  reinicia(requisições [indice]);
20                  reinicia(respostas [indice]);
21              end
22          end
23          else
24              reinicia(requisições [indice]);
25              reinicia(respostas [indice]);
26          end
27      end
28  end
```

Em seguida, inicia-se a parte em que se testa diversas condições para verificar se a resposta recebida pode pertencer a um ataque. A primeira condição verifica se o intervalo entre duas respostas, direcionadas para o mesmo destino (a vítima), é suficientemente pequeno para ser caracterizado como um ataque (linha 15). Vale lembrar que em ataques de

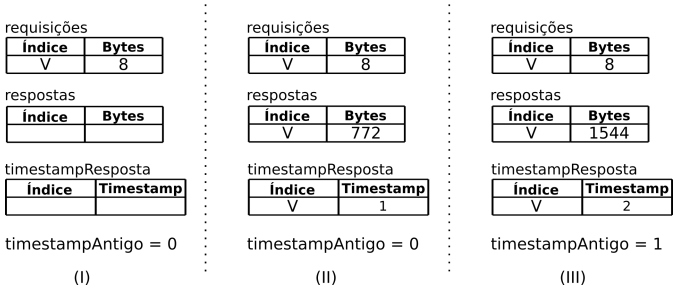
negação de serviço, pacotes devem chegar em intervalos de tempo relativamente pequenos para que haja exaustão da largura de banda da vítima. Então, caso o resultado dessa condição seja falso, está-se lidando com respostas que chegaram distantes uma da outra, o que não caracteriza um ataque. Por isso, reinicia-se os valores dos contadores associados a essa vítima (linhas 24 e 25). Agora, caso o condicional se apresente verdadeiro, de maneira similar ao caso da requisição, deve-se somar o tamanho da mensagem no registrador que contabiliza o número de *bytes* recebidos através de respostas (linha 16). Na sequência, realiza-se outro teste para averiguar a diferença entre os *bytes* de requisições e respostas (linha 17). Caso essa diferença resulte em um valor maior do que o aceitável (`BYTES_THERSHOLD`), ou seja, há muito mais *bytes* provenientes de respostas do que de requisições, caracteriza-se um ataque de negação de serviço com amplificação. O dispositivo, então, pode executar alguma ação para mitigar o evento, como, por exemplo, enviar uma mensagem de alerta para o controlador da rede. Para que o algoritmo não fique acumulando dados indefinidamente, após a detecção de um ataque são zerados os contadores de *bytes* referentes à essa vítima.

Exemplificando a detecção próxima ao servidor. O exemplo apresentado nas Figuras 3.3a e 3.3b busca ilustrar as informações manipuladas pelo Algoritmo 1 durante um ataque. Por simplicidade, usa-se os valores: `TS_THRESHOLD = 5` e `BYTES_THRESHOLD = 1000`. Os contadores e tabelas utilizados encontram-se inicialmente zerados.

Figura 3.3: Exemplo de uma situação de ataque.



(a) Requisições originadas em A são respondidas para V.



(b) Exemplo de preenchimento das informações mantidas pelo dispositivo.

Fonte: Elaborado pelo autor.

Primeiramente, o atacante A requisita a lista de monitoração do amplificador AMP, forjando seu endereço de origem com o endereço da vítima V. O *switch* identifica essa mensagem e insere no contador de *bytes* de requisições para V o valor 8, que é a quantidade padrão de *bytes* contida em um pedido `monlist` (Figura 3.3b - (I)). Ao receber a solicitação, a lista de monitoração armazenada pelo amplificador é relativamente grande, de forma que ele necessita dividir sua resposta em duas mensagens de 772 *bytes* cada. No momento em que a primeira dessas respostas chega no *switch*, com *timestamp* 1, insere-se uma entrada na tabela que armazena o momento da última resposta analisada. A variável `timestampAntigo` recebe zero, pois ainda não havia nenhuma entrada na tabela. Assim, o condicional da linha 15 apresenta-se verdadeiro, pois a diferença entre *timestamp* atual e antigo resulta em um valor menor que `TS_THRESHOLD`. Após, a tabela que armazena a quantidade de *bytes* oriundos de respostas é incrementada com os 772 *bytes* da mesma. O resultado do teste presente na linha 17 é negativo pois 772 - 8 = 764 e esse valor é menor que `BYTES_THRESHOLD`. Os dados mantidos pelo dispositivo ao final dessas ações podem ser observados na Figura 3.3b - (II).

A segunda resposta atinge o *switch* no *timestamp* 2. De maneira semelhante ao pacote anterior, atualiza-se os *timestamps* e a tabela de *bytes* de respostas como ilustra a Figura 3.3b - (III). Contudo, nessa caso a verificação da linha 17 resulta em positiva, pois 1544 - 8 = 1536, e esse valor é maior que o limite `BYTES_THRESHOLD`. Esse resultado, então, ocasiona na identificação de um ataque cuja vítima é V. Por fim, conforme o algoritmo, os contadores de requisições e respostas são reiniciados.

3.3 Detecção Próxima à Vítima

Esta subseção visa detalhar o algoritmo projetado para o Cenário 2. De maneira análoga à situação anterior, os dados de entrada continuam sendo os campos dos cabeçalhos extraídos do pacote.

Mecanismo proposto. Em suma, esse algoritmo analisa a assimetria entre pacotes NTP `monlist`, e não *bytes*, visto que nesse contexto um ataque se caracteriza por enviar diversas respostas que não foram requisitadas pela vítima. Para isso, mantém-se, para cada vítima, um contador de mensagens, sejam elas requisições ou respostas. Isto é, caso a mensagem que está sendo processada seja uma requisição, incrementa-se o contador associado em uma unidade. Agora, caso se trate de uma resposta, decrementa-se o contador

da mesma maneira. Assim, a medida que ocorram mais respostas do que requisições, o contador terá o seu valor cada vez menor e, baseado nisso, um ataque pode ser identificado caso esse valor ultrapasse um certo limite pré definido. De maneira análoga ao cenário anterior, o mecanismo de verificação de *timestamps* também é utilizado.

Algoritmo 2: Algoritmo para o detector localizado na rede da vítima.

```
    entrada: header: refere-se ao cabeçalho do pacote que acabou de ingressar no
             pipeline do switch, após ter passado pelo parser.
 1  for indice = 0 to tamanho(mensagens) do
 2      mensagens [indice] = 0;
 3      timestampResposta [indice] = 0;
 4  end
 5  if cabeçalho.NTP.Valido() then
 6      if cabeçalho.NTP.requestCode == REQUISICAO_MONLIST then
 7          indice = hash(cabeçalho.IPorigem);
 8          mensagens [indice] = mensagens [indice] + 1;
 9      end
10      if cabeçalho.NTP.requestCode == RESPOSTA_MONLIST then
11          indice = hash(cabeçalho.IPdestino);
12          timestampAntigo = timestampResposta [indice];
13          timestampResposta [indice] = timestampAtual;
14          if timestampResposta [indice] − timestampAntigo <
             TS_THRESHOLD then
15              mensagens [indice] = mensagens [indice] − 1;
16              if mensagens [indice] < MESSAGES_THRESHOLD then
17                  Ataque Detectado;
18                  reinicia(mensagens [indice]);
19              end
20          end
21          else
22              reinicia(mensagens [indice]);
23          end
24      end
25  end
```

Apresenta-se o Algoritmo 2 para melhor compreensão do mecanismo. Note que o mesmo é muito semelhante com o proposto no cenário anterior, diferenciando-se pela ação executada dependendo da mensagem NTP recebida. Isto é, ao invés de se somar os *bytes* das mensagens, incrementa-se ou decrementa-se o contador, conforma as linhas 8 e 15. Outra mudança ocorre no condicional da linha 16, que agora verifica se o contador se encontra abaixo de um determinado valor limite (`MESSAGES_THRESHOLD`). Por fim, caso ocorra a situação em que duas respostas chegam em um intervalo de tempo em que não se caracteriza mais um ataque DDoS, a única estrutura a ser reiniciada é o contador de mensagens (linha 22).

Como mencionado anteriormente, esse cenário possui a restrição de que não podem haver atacantes conectados ao mesmo dispositivo que a vítima. Isso se deve ao fato de que o algoritmo foi proposto para detectar respostas para as quais não existem as equivalentes requisições. Explicando, quando o atacante disparasse requisições forjadas para o amplificador, o *switch* incrementaria o contador relativo àquele endereço IP de origem. No momento em que as respectivas respostas fossem recebidas, o equipamento decrementaria esse mesmo contador, de forma que ele nunca chegaria a um valor pequeno o suficiente para caracterizar um ataque.

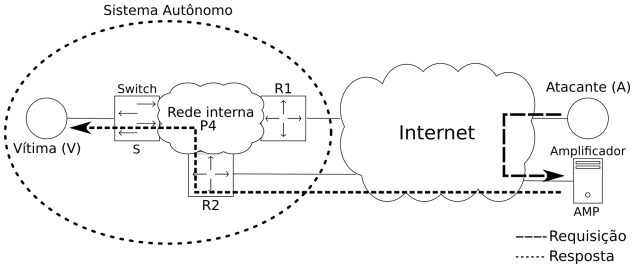
A exceção recém mencionada pode ser mitigada implementando, no dispositivo de detecção, um mecanismo de associação entre a porta de entrada de um pacote e seu endereço, de forma que caso seja identificada uma requisição cujo IP de origem não corresponda à porta associada, essa mensagem pode ser considerada como atividade maliciosa. Porém, essa abordagem não pôde ser desenvolvida devido a limitações de tempo.

Exemplificando a detecção próxima à vítima. O exemplo apresentado nas Figuras 3.4a e 3.4b busca ilustrar as informações manipuladas pelo Algoritmo 2 durante um ataque. Utiliza-se os valores: `TS_THRESHOLD = 5` e `MESSAGES_THRESHOLD = -1` por simplicidade. Os contadores e tabelas utilizados encontram-se inicialmente zerados.

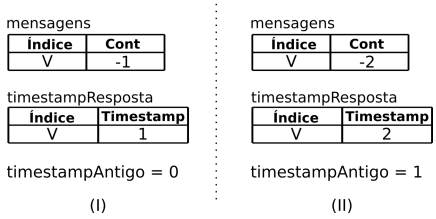
Inicialmente, o atacante A requisita a lista de monitoração do amplificador AMP, falsificando o endereço IP de origem dessa mensagem com o endereço da vítima V. Da mesma forma que no cenário anterior, o amplificador encaminha sua resposta para V em dois pacotes, devido ao tamanho da lista armazenada. Essa troca de mensagens está identificada na Figura 3.4a pelas linhas tracejadas. O *switch* S identifica o primeiro pacote de resposta no *timestamp* 1. Nesse momento, ele cria uma entrada na tabela que armazena o momento da última resposta com o valor 1. A variável `timestampAntigo` recebe zero, pois ainda não havia nenhuma entrada nessa tabela. Assim, o condicional da linha 14 apresenta-se verdadeiro, pois a diferença entre *timestamp* atual e antigo resulta em um valor menor que `TS_THRESHOLD`. Como resultado, o contador de mensagens é decrementado e as informações armazenadas ficam como exemplificadas na Figura 3.4b - (I). Logo após, o teste da linha 16 resulta negativo, visto que o contador de mensagens não é menor que `MESSAGES_THRESHOLD`.

A segunda resposta atinge o *switch* no *timestamp* 2. Da mesma maneira que a resposta anterior, atualiza-se os *timestamps* e o contador de mensagens, resultando em uma situação como a da Figura 3.4b - (II). Nesse caso, os condicionais da linha 14 e 16 resul-

Figura 3.4: Exemplo de uma situação de ataque.



(a) Requisições originadas em A são respondidas para V.



(b) Exemplo de preenchimento das informações mantidas pelo dispositivo.

Fonte: Elaborado pelo autor.

tam positivo, pois -2 é menor que o limite `MESSAGES_THRESHOLD`. Em consequência desse resultado, um ataque é detectado para a vítima V e o contador de mensagens reiniciado.

3.4 Discussão

Para finalizar a proposta, esta seção tem por objetivo discutir e esclarecer alguns tópicos deixados em aberto com relação às estratégias apresentadas. Argumenta-se, primeiramente, sobre o intervalo de tempo entre respostas considerado aceitável. Trabalhar com janelas de tempo grandes demais significa alocar e manter recursos computacionais por longos períodos de tempo. Isso pode acabar comprometendo o dispositivo, de forma que ele próprio possa ser usado como alvo de um ataque. Além disso, a escolha de um intervalo limite elevado pode acarretar em um acréscimo significativo no número de falsos positivos (tráfego legítimo considerado como ataque) observados. Por exemplo, pode-se citar um usuário que executa requisições legítimas frequentemente porém sem intuito malicioso. Nesse caso, se a janela estiver definida em um valor alto demais, essas requisições podem ser identificadas como maliciosas. Por outro lado, caso esse limiar seja pequeno demais, o atacante pode burlar essa janela de tempo, fazendo com que respostas cheguem em intervalos maiores que o limite definido e ainda assegurando a eficiência do ataque. Tal ocasionaria um aumento no número de falsos negativos (tráfego malicioso

considerado como legítimo).

Outro ponto a ser discutido é o limite aceitável para a diferença de *bytes* entre respostas e requisições. Em um cenário em que esse valor seja pequeno demais, usuários legítimos podem ser identificados como maliciosos dependendo do tamanho da resposta. De maneira oposta, a definição um valor relativamente alto pode ocasionar a ocorrência de um ataque não detectado (falso negativo).

Diante das questões apresentadas, acredita-se que os quesitos abordados nessa seção deveriam ser parametrizáveis, pois cada infraestrutura de rede possui um determinado potencial em termos de capacidade agregada, de tal modo que determinar quão frequente esses tipos de situações podem ocorrer depende muito dessas capacidades. Em uma rede de pequeno porte, a margem de tolerância em termos de repetidas solicitações por um determinado comando pode possuir um valor comparativamente menor do que para uma rede com capacidade agregada mais elevada. Além disso, chama-se atenção para o fato de que em alguns casos o mecanismo pode não se tratar de detectar ou não ataques. Pode ser visto como uma maneira de lidar com comportamentos ditos aceitáveis ou não baseada nas características da infraestrutura. Por exemplo, qual seria a janela de tempo a ser usada em uma rede na qual requisições `monlist` são frequentes, porém não devem ser tratadas como maliciosas? Portanto, no contexto deste trabalho, deixa-se em aberto a proposição de mecanismos adaptativos para definição de limiares.

4 IMPLEMENTAÇÃO DOS MECANISMOS UTILIZANDO P4

É importante recapitular que dentre os objetivos do trabalho, propõe-se a implementação dos mecanismos propostos no contexto da linguagem P4. O grande desafio deste capítulo residiu em tirar proveito de construtores limitados oferecidos pela linguagem e explorá-los na direção de modelar os mecanismos de detecção em cima de planos de dados programáveis. Atualmente, a linguagem possui duas versões: $P4_{14}$ e $P4_{16}$. Por $P4_{16}$ ser a mais recente, a implementação dos mecanismos ocorreu na versão 1.0.0 da mesma. Nesse capítulo, expõe-se uma visão geral das estruturas de dados utilizadas e, em seguida, detalham-se as mesmas para cada um dos mecanismos exibidos no capítulo anterior.

4.1 Estuturas de Dados Empregadas

Como mencionado na Seção 2.2, o processamento de pacotes, em um programa P4, é implementado através de tabelas do tipo *match-action*. Usualmente, essas tabelas são populadas e configuradas pelo plano de controle, como ilustra a Figura 2.5. Contudo, quem define as chaves e as ações possíveis de serem tomadas é o plano de dados. Pelo fato dos mecanismos terem sido propostos para executar inteiramente no plano de dados, foi utilizada uma função da linguagem P4 que permite definir uma ação padrão para qualquer entrada, sem a necessidade da comunicação com plano de controle. Assim, todas as tabelas *match-action* implementadas possuem apenas uma entrada e uma ação padrão associada, como representado na Figura 4.1. No exemplo dessa figura, todas as repostas NTP `monlist` que ingressarem no dispositivo serão direcionados para essa tabela. A coluna chave possui uma única entrada, *DEFAULT*, indicando que a ação *atualizaTimestamp* é executada independente dos atributos do pacote ingressante. Como será detalhado mais adiante, os mecanismos propostos baseiam-se no encadeamento de tabelas desse tipo.

Além das referidas tabelas, foram utilizadas estruturas da linguagem chamadas registradores. Essas estruturam representam dados globais perduráveis enquanto o dispo-

Figura 4.1: Tabela *match-action* de tamanho unitário.

`setaTimestampResposta`

Chave	Ação
DEFAULT	atualizaTimestamp

Fonte: Elaborado pelo autor.

sitivo estiver ligado. Para os mecanismos implementados, eles armazenam dados como: *bytes* de requisições, *bytes* de respostas, contadores de mensagens e *timestamps*.

Os registradores utilizados para armazenar quantidade de *bytes* possuem N posições, capazes de armazenar valores de 32 bits. Os registradores definidos para memorizar *timestamps* possuem o mesmo número de posições, porém com largura de dados de 48 bits. Cada índice desses registradores representa uma vítima, cujo valor é calculado através de uma função *hash* utilizando o endereço IP da mesma. Caso seja uma requisição, o índice é calculado utilizando o endereço de origem da mensagem. Caso seja uma resposta, ele é calculado usando o endereço de destino. O valor resultante dessa operação está limitado ao intervalo entre 0 e $N - 1$. Para estimar a quantidade de memória usada pelo arranjo de registradores de cada categoria, pode ser usado o cálculo: $Tamanho = N * L$ bits, onde L representa a largura de dados e N identifica o número máximo de posições do registrador.

Do mesmo modo, foram utilizadas estruturas de metadados que, diferentemente dos registradores, perduram apenas enquanto um pacote estiver ativo no *pipeline* do *switch*. Esses metadados são utilizados em duas situações: primeiro, para armazenar dados lidos de registradores, visto que não é possível utilizá-los diretamente para testar condições; segundo, para executar as diversas operações aritméticas necessárias, tais como diferença entre *bytes* ou diferença entre *timestamps*. Os parágrafos que se seguem buscam esclarecer em detalhes os diferentes tipos de estruturas utilizados por cada algoritmo proposto no Capítulo 3.

Algoritmo 1 - Detecção próxima ao serviço comprometido. Utilizando a Figura 4.4, ilustra-se as estruturas de dados implementadas no mecanismo de detecção próximo ao serviço comprometido, correspondente ao Algoritmo 1 da Seção 3.2. A seção (I) da figura representa as estruturas empregadas quando a mensagem processada é uma requisição, enquanto a seção (II) representa uma resposta. Ambas seções retratam as tabelas, e suas respectivas ações, aplicadas aos pacotes durante sua passagem pelo *switch*.

Para o Algoritmo 1, foram definidas quatro tabelas *match-action*, conforme sumariza a Tabela 4.1. A primeira coluna identifica o nome de cada uma delas, e a segunda descreve a função das mesmas. A primeira dessas tabelas pode ser visualizada na Figura 4.4 - (I), enquanto as demais estão identificadas na Figura 4.4 - (II).

As variáveis de metadados utilizadas estão identificadas nos retângulos mais superiores de ambas as seções da Figura 4.4, representando que esses dados estão acessíveis

Tabela 4.1: Tabelas *match-action* utilizadas para o Algoritmo 1.

Nome	**Descrição**
`setaBytesRequisição`	Responsável por somar os *bytes* oriundos de uma requisição.
`setaBytesResposta`	Responsável por somar o número de *bytes* provenientes de uma resposta.
`setaTimestampResposta`	Responsável por atualizar o *timestamp* da última resposta recebida, assim como o metadado que contém o *timestamp* antigo.
`reiniciaRegistrador`	Responsável por reiniciar o número de *bytes* de requisições e respostas.

apenas entre a entrada e saída de cada pacote. A Tabela 4.2 sumariza a função de dessas variáveis.

Tabela 4.2: Metadados utilizados para o Algoritmo 1.

Nome	**Descrição**
`indice`	Variável utilizada para armazenar o resultado da operação de *hash*. Possui 32 bits.
`bytesReq`	Variável utilizada para armazenamento do valor lido do registrador `requisições`. Possui 32 bits.
`bytesResp`	Variável utilizada para armazenamento do valor lido do registrador `respostas`. Possui 32 bits.
`timestampAntigo`	Variável utilizada para armazenar o valor lido do registrador `timestampResposta`, antes do mesmo ser atualizado. Possui 48 bits.
`timestampAtual`	Variável utilizada para armazenar o *timestamp* do pacote ingressante. Possui 48 bits.

A seção (III) dessa mesma figura demonstra os registradores utilizados nesse mecanismo, representando uma estrutura acessível e persistente entre os diferentes pacotes processados. Cada um dos registradores implementados está enumerado na Tabela 4.3. A primeira coluna dessa tabela identifica o nome de cada um deles, enquanto a segunda coluna descreve o tipo de dado que os mesmos armazenam.

Tabela 4.3: Registradores utilizados para o Algoritmo 1.

Nome	**Descrição**
`requisições[N]`	Responsável por armazenar os *bytes* recebidos por requisições `monlist`.
`respostas[N]`	Armazena os *bytes* recebidos através de respostas `monlist`.
`timestampResposta[N]`	Memoriza o *timestamp* da última resposta `monlist` processada.

A Figura 4.2 ilustra o programa de controle, implementado em P4, para o Algo-

ritmo 1. Com o auxílio da mesma, demonstra-se o fluxo dos dados dentro do dispositivo de encaminhamento para uma requisição `monlist`. De acordo com o código, após a identificação da requisição, a tabela aplicada ao pacote é a `setaBytesRequisição`, cuja ação associada é a `somaBytesReq`, conforme a Figura 4.4 - (I). Nessa ação, representada no retângulo logo abaixo da tabela, primeiramente calcula-se o índice da vítima, armazenando-o no metadado `indice`. Em seguida, executa-se uma operação de leitura no registrador `requisições[indice]`, memorizando o resultado dessa operação em `bytesReq`. Logo após, soma-se a essa variável a quantidade de *bytes* de uma requisição, para depois escrever o valor resultante de volta no registrador. Por fim, após executada a ação, o programa de controle retoma a execução, e direciona o pacote para saída do dispositivo.

Figura 4.2: Programa de controle P4 para o Algoritmo 1.

```
if(header.ntpMonlist.isValid()) {
    if (header.ntpMonlist.requestCode == REQUISICAO_MONLIST) {
        setaBytesRequisicao.apply();
    } else if (header.ntpMonlist.requestCode == RESPOSTA_MONLIST) {
        setaTimestampResposta.apply();
        /* Teste timestamps */
        if (metadado.timestampAtual - metadado.timestampAntigo < TS_THRESHOLD) {
            /* Teste bytes */
            if (metadado.bytesResp - metadado.bytesReq > BYTES_THRESHOLD) {
                ataqueDetectado.apply();
            }
        } else {
            reiniciaRegistrador.apply();
        }
    }
}
```

Fonte: Elaborado pelo autor.

Algoritmo 2 - Detecção próxima à vítima. Da mesma maneira que para o algoritmo anterior, utilizando a Figura 4.5, exemplifica-se as estruturas utilizadas no mecanismo de detecção implementado próximo à vítima, apresentado no Algoritmo 2 da Seção 3.3. Para esse mecanismo, também foram definidas quatro tabelas *match-action*, de acordo com a Tabela 4.4.

Os registradores implementados se diferenciam dos mencionados no mecanismo anterior pela substituição dos contadores de *bytes* por um contador de mensagens: `mensagens[N]`. Além disso, o registrador `timestampResposta` continua com a mesma função, como sumariza a Tabela 4.5. Com relação aos metadados utilizados, mantém-se a maioria dos mencionados anteriormente, substituindo `bytesReq` e `bytesResp` por `mensagensCont`, cuja função é armazenar o valor lido do registrador `mensagens`.

Para exemplificar o fluxo desse algoritmo, cujo programa de controle está repre-

Tabela 4.4: Tabelas *match-action* utilizadas para o Algoritmo 2.

Nome	**Descrição**
`incrementaContador`	Responsável por adicionar uma unidade ao contador de mensagens.
`decrementaContador`	Responsável por decrementar uma unidade do contador de mensagens.
`setaTimestampResposta`	Responsável por atualizar o *timestamp* da última resposta recebida, assim como o metadado que contém o *timestamp* antigo.
`reiniciaRegistrador`	Responsável por reiniciar o contador de mensagens.

Tabela 4.5: Registradores utilizados para o Algoritmo 2.

Nome	**Descrição**
`requisições[N]`	Responsável por armazenar os *bytes* recebidos por requisições `monlist`.
`respostas[N]`	Armazena os *bytes* recebidos através de respostas `monlist`.
`timestampResposta[N]`	Memoriza o *timestamp* da última resposta `monlist` processada.

sentado na Figura 4.3, usa-se o exemplo em que uma resposta ingressa no dispositivo espaçada de tal forma da resposta anterior que não caracteriza um ataque. Conforme a Figura 4.5 - (II), a primeira tabela aplicada a esse pacote é a `setaTimestampResposta`. Na ação associada, calcula-se o índice da vítima da mesma maneira que explicado anteriormente. Logo após, o *timestamp* de ingresso do pacote atual é obtido através de uma estrutura chamada `ingress_global_timestamp`, fornecida pelo próprio equipamento, e armazenado na variável `timestampAtual`. Em seguida, copia-se para o metadado `timestampAntigo` o número contido no registrador `timestampRespostas` para depois atualizar o valor do mesmo. A partir da verificação entre *timestamps*, toma-se o caminho inferior em que a tabela `reiniciaRegistrador` executa a ação `zeraMensagens`. Nessa ação, o valor do contador de mensagens para a vítima é zerado e o pacote é encaminhado para a saída do equipamento.

Figura 4.3: Programa de controle P4 para o Algoritmo 2.

```
if(header.ntpMonlist.isValid()) {
    if (header.ntpMonlist.requestCode == REQUISICAO_MONLIST) {
        incrementaContador.apply();
    } else if (header.ntpMonlist.requestCode == RESPOSTA_MONLIST) {
        setaTimestampResposta.apply();
        /* Teste timestamps */
        if (metadado.timestampAtual - metadado.timestampAntigo < TS_THRESHOLD) {
            /* Teste bytes */
            if (metadado.mensagensCont < MESSAGES_THRESHOLD) {
                ataqueDetectado.apply();
            }
        } else {
            reiniciaRegistrador.apply();
        }
    }
}
```

Fonte: Elaborado pelo autor.

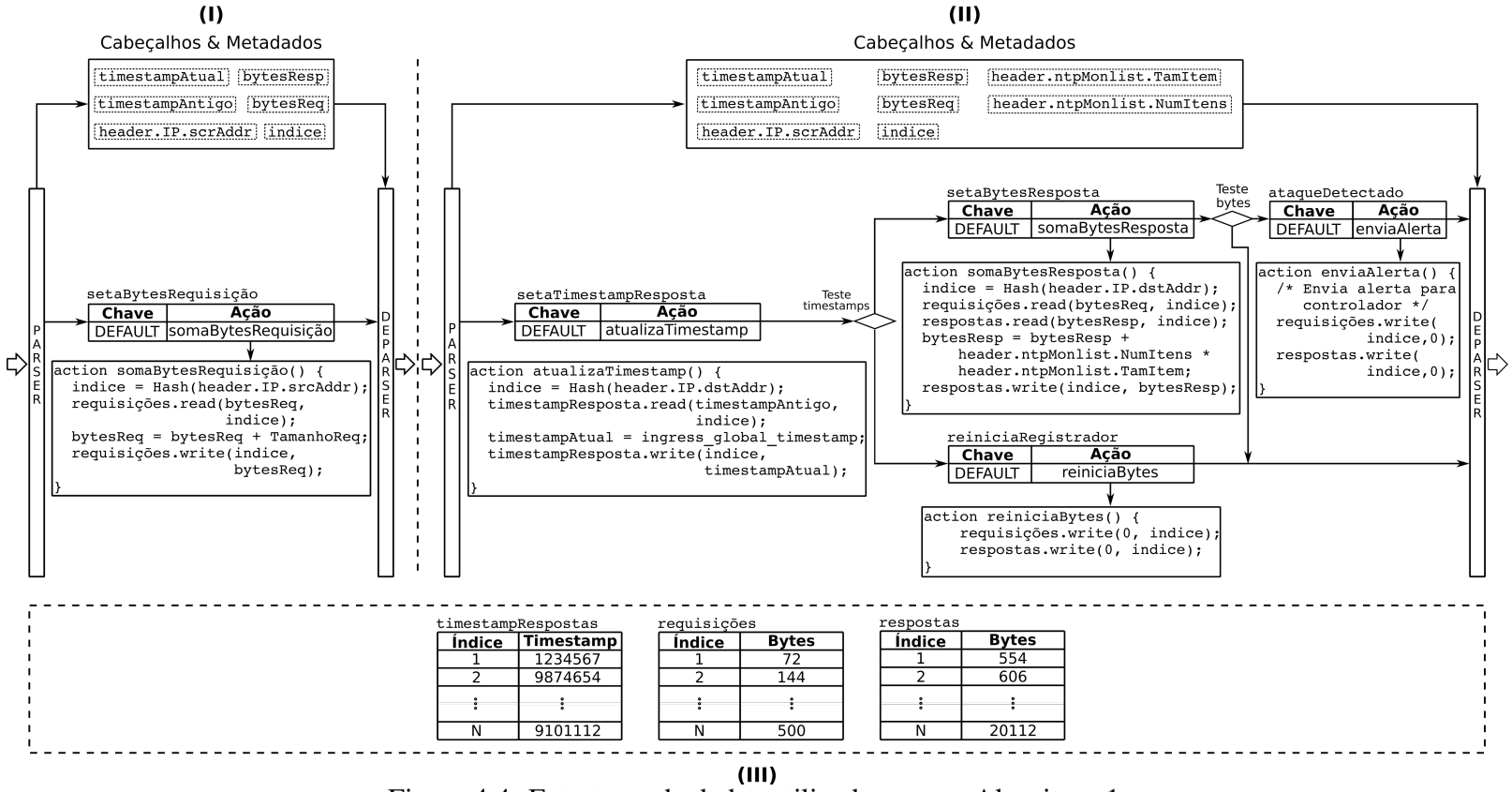


Figura 4.4: Estruturas de dados utilizadas para o Algoritmo 1.
Fonte: Elaborado pelo autor.

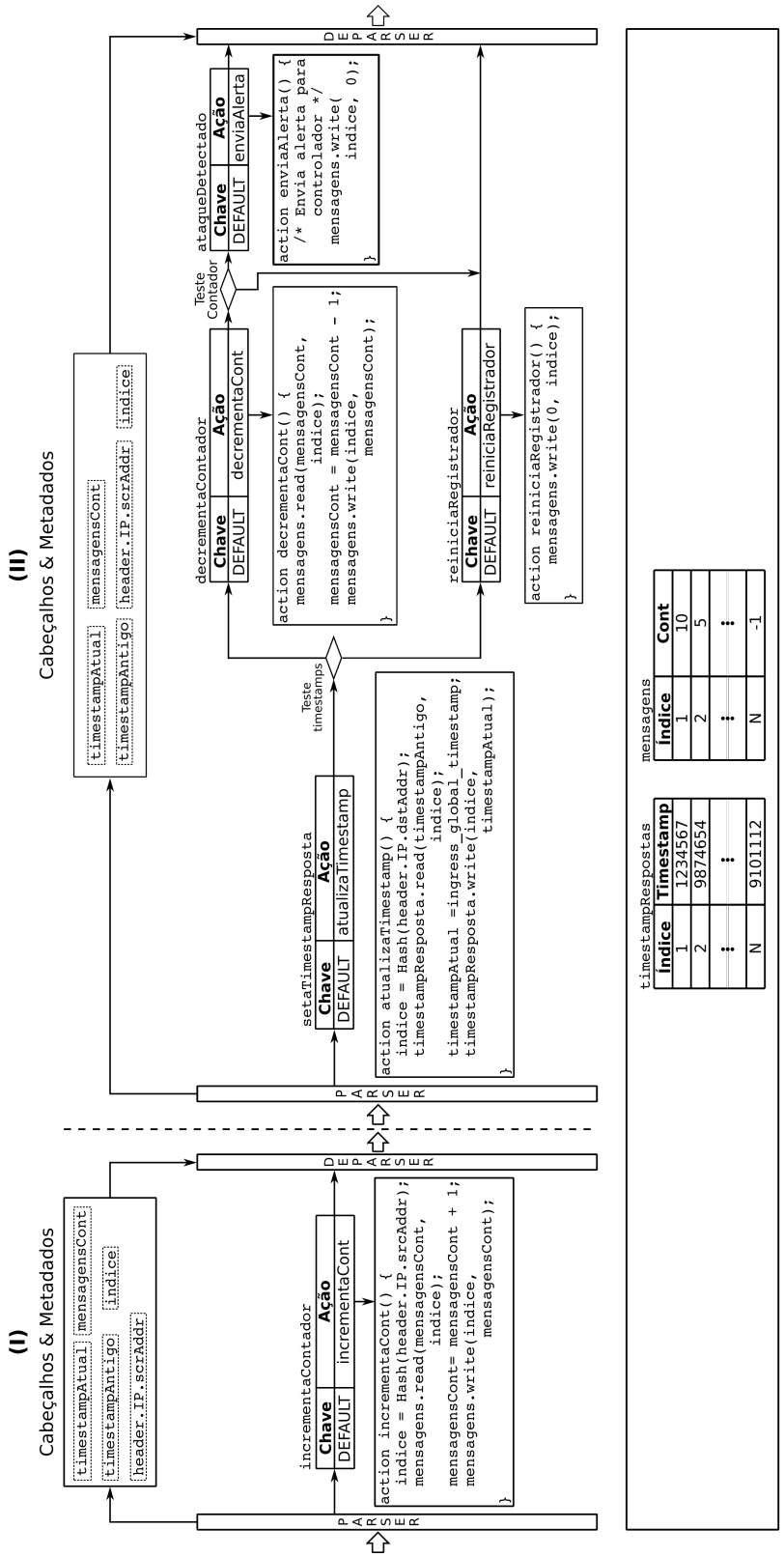


Figura 4.5: Estruturas de dados utilizadas para o Algoritmo 2.
Fonte: Elaborado pelo autor.

5 AVALIAÇÃO DOS MECANISMOS DE DETECÇÃO

Este capítulo apresenta informações relacionadas à avaliação de desempenho dos mecanismos propostos. Na Seção 5.1, realiza-se a descrição das tecnologias utilizadas e as métricas consideradas para análise. Por fim, a Seção 5.2 apresenta e discute os resultados obtidos a partir dos diferentes cenários nos quais os mecanismos se inserem.

5.1 Ambiente de Experimentação e Métricas

Utilizou-se o emulador *Mininet* (TEAM, 2012) por se tratar do ambiente usualmente empregado para avaliações de Redes Definidas por Software. Além disso, o mesmo possui uma interface capaz de emular o comportamento de dispositivos P4 através do *Behavioral Model*, desenvolvido pelos idealizadores da linguagem P4 (P4 LANGUAGE CONSORTIUM, 2017).

Pela dificuldade de se obter traços de ataques de amplificação como os avaliados neste trabalho, optou-se por simular o comportamento de clientes e servidores NTP utilizando a biblioteca de manipulação de pacotes *Scapy*, desenvolvida na linguagem de programação *Python*. Com relação aos resultados obtidos, cada experimento foi realizado, no mínimo, dez vezes e as quantidades apresentadas representam a média dessas medições.

Avalia-se a eficácia dos mecanismos implementados através das três métricas enumeradas a seguir.

- *Falsos positivos*: refere-se a clientes que executaram requisições legítimas, porém foram reportados como atacantes.
- *Falsos negativos*: trata-se de situações maliciosas tratadas como legítimas.
- *Tempo de detecção*: versa sobre o tempo decorrido até a detecção de um ataque.

Como uma análise complementar para os casos de falsos positivos e falsos negativos, avalia-se também a acurácia dos mecanismos através das métricas de verdadeiros positivos e verdadeiros negativos. A primeira refere-se ao sucesso na detecção de ataques, enquanto a segunda representa o sucesso na identificação de clientes legítimos.

5.2 Resultados

Os resultados apresentados nesta seção estão organizados da seguinte maneira: primeiramente, são avaliados os números de falsos positivos e de falsos negativos. Na sequência, apresenta-se a análise do tempo decorrido até a detecção de um ataque. Em cada análise, apresenta-se primeiro os resultados obtidos para o mecanismo localizado próximo do serviço comprometido (Seção 3.2), seguido pela avaliação dos resultados para o mecanismo estabelecido nas adjacências da vítima (Seção 3.3).

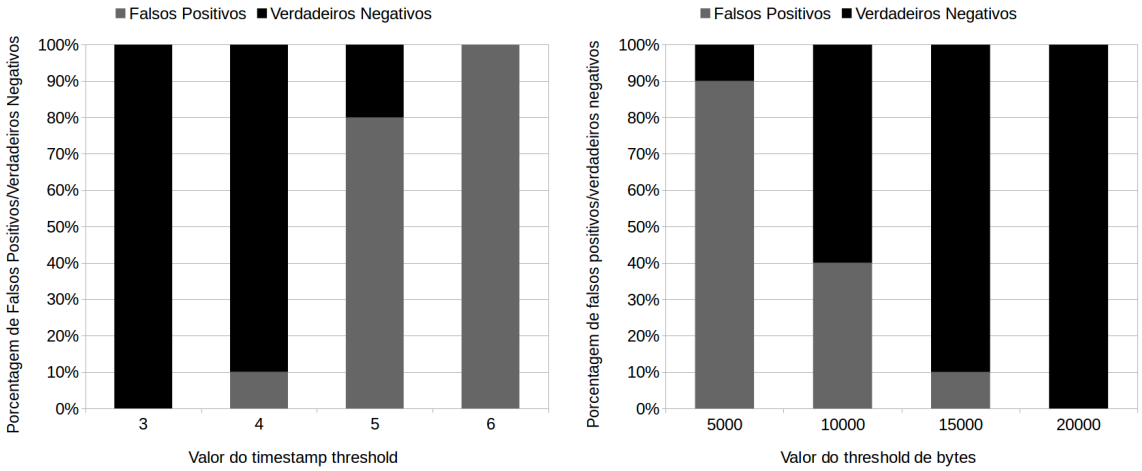
5.2.1 Falsos Positivos

Mecanismo próximo ao serviço comprometido. Para a avaliação do número de falsos positivos, utilizou-se um cenário em que dez clientes, sem intenções maliciosas, requisitam a `monlist` de um servidor NTP em intervalos aleatórios entre dois e cinco segundos. Os registradores de dados P4 possuem 16 entradas, o fator de amplificação do servidor é 540 e o tempo de duração da análise é de 60 segundos. Vale ressaltar que nessas avaliações, a situação ideal seria um total de 0% de falsos positivos.

Com base nesse cenário, a Figura 5.1 apresenta o impacto que diferentes valores para *timestamp threshold* e *bytes threshold* causam na acurácia da detecção. A Figura 5.1a ilustra a porcentagem de falsos positivos e verdadeiros negativos em função do valor empregado para o parâmetro *timestamp threshold*, utilizando *bytes threshold* fixo em 30.000. A primeira medição da figura, em que *timestamp threshold* possui valor três segundos, representa o cenário base em que todos os clientes são identificados como legítimos e nenhum ataque é detectado (100% de verdadeiros negativos).

De maneira geral, quanto maior for o *timestamp threshold*, mais pacotes de resposta serão contabilizados nesse intervalo, e consequentemente, maior a chance desses *bytes* de resposta ultrapassarem o limite imposto pelo *threshold* de *bytes*. Assim, é de se esperar que o aumento do intervalo definido pelo *timestamp threshold* cause um aumento no número de falsos positivos. A Figura 5.1a demonstra exatamente essa tendência. Como pode-se observar, quando o valor de *timestamp threshold* é quatro, apenas um dentre os dez clientes foi erroneamente identificado como atacante, contabilizando um total de 10% de falsos positivos. Esse valor aumenta em cada uma das medições seguintes, chegando a atingir 100% na última aferição. Dessa forma, pode-se concluir que, para uma rede com características semelhantes a desse cenário (clientes requisitando `monlists` com

Figura 5.1: Resultados obtidos a partir da avaliação do número de falsos positivos observados para o mecanismo próximo ao servidor NTP.



(a) Falsos positivos com relação à variação do valor de *timestamp threshold*.

(b) Falsos positivos com relação à variação do valor de *bytes threshold*.

frequência entre dois e cinco segundos, fator de amplificação do servidor igual a 540, *etc*), o melhor valor a ser escolhido para *timestamp threshold* é três segundos.

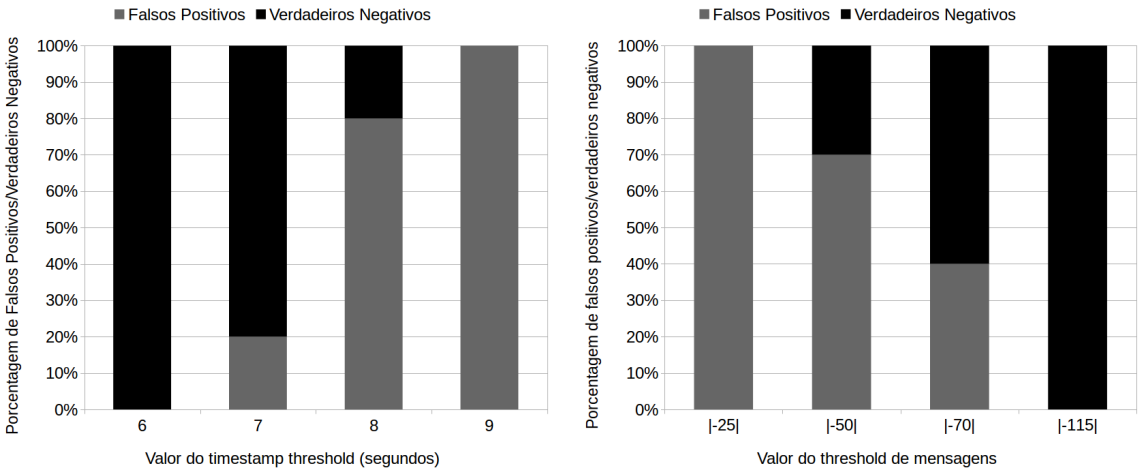
De forma análoga, a Figura 5.1b apresenta a quantidade de falsos positivos e verdadeiros negativos em função do valor utilizado para o parâmetro *bytes threshold*, fixando *timestamp threshold* em três segundos. Na primeira medição, em que *bytes threshold* = 5.000, 90% dos clientes ativos foram reportados como ataques. Contudo, a figura mostra que o aumento desse parâmetro causa uma diminuição no número de falsos positivos observados, de forma que essa porcentagem atingiu o valor de 0% na última medição feita. Isso se deve ao fato de que, com um *threshold* de *bytes* pequeno, um baixo número de respostas é suficiente para que esse limite seja atingido e, consequentemente, um ataque reportado.

Em resumo, as avaliações apresentadas sugerem que a acurácia do mecanismo pode ser influenciada pelos valores escolhidos para *timestamp threshold* e também para o *threshold* de *bytes*. Dado o cenário apresentado, a melhor configuração para esses valores seria: *timestamp threshold* = 3 segundos e *bytes threshold* = 20.000 *bytes*.

Mecanismo próximo à vítima. Como mencionado na Seção 3.3, em que se propõe o mecanismo localizado próximo à vítima, esse cenário se caracteriza por uma menor taxa de requisições pela `monlist` de um servidor NTP, comparado com o contexto anterior. Assim, para a avaliação desse mecanismo foi utilizada uma situação semelhante (dez clientes legítimos, sem ataques, servidor com fator de amplificação 540), porém as

requisições são feitas em intervalos maiores: entre seis e nove segundos.

Figura 5.2: Resultados obtidos a partir da avaliação do número de falsos positivos observados para o mecanismo localizado próximo à vítima.



(a) Falsos positivos com relação à variação do valor de *timestamp threshold*.

(b) Falsos positivos com relação à variação do valor de *messages threshold*.

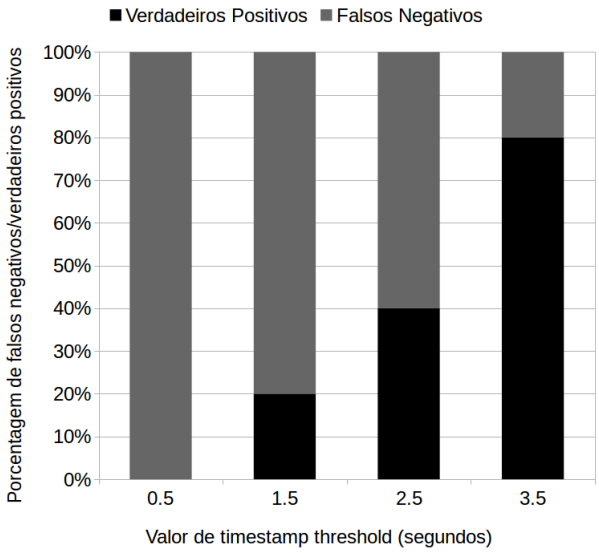
A Figura 5.2a ilustra os resultados obtidos considerando a variação do parâmetro *timestamp threshold*. Da mesma forma que para o mecanismo anterior, variar a janela de tempo causa um aumento no número de falsos positivos. Nesses experimentos, foi necessário utilizar valores mais altos para tal parâmetro, pois requisições não são tão frequentes. Nota-se que para um *timestamp threshold* de seis segundos, tem-se 100% de acerto em não detectar clientes legítimos como atacantes. Contudo, os valores observados para essa métrica diminui com o aumento da janela de tempo utilizada, chegando até o valor de 0% quando foi utilizado o valor de nove segundos.

Já a Figura 5.2b apresenta como a variação do parâmetro *messages threshold* afeta a porcentagem de verdadeiros negativos identificados. O eixo horizontal do gráfico representa o aumento do módulo do valor utilizado para o *threshold* de mensagens, ou seja, o aumento da diferença necessária entre respostas e requisições NTP `monlist` para que um ataque seja detectado. Nessas aferições, o valor de *timestamp threshold* foi mantido em sete segundos. Pode-se perceber que quanto maior o módulo do *threshold* de mensagens, maior a porcentagem de acerto do mecanismo. Quando se utiliza um valor muito pequeno (|-25|), todas requsições são identificadas como ataques. Entretanto, quando esse valor chega a |-115|, nenhuma requisição é tratada como ataque e tem-se 100% de verdadeiros negativos.

5.2.2 Falsos Negativos

Mecanismo próximo ao serviço comprometido. Para a avaliação da porcentagem de falsos negativos, utilizou-se um cenário em que estão ocorrendo cinco ataques simultaneamente (para vítimas diferentes), e nenhuma requisição legítima. Cada um desses ataques possui uma intensidade diferente entre: 34 kbps, 17 kbps, 13 kbps, 11 kbps e 6 kbps. De forma semelhante à análise anterior, os registradores de dados P4 possuem 16 entradas, o fator de amplificação do servidor é de 540 e o tempo de duração da análise é de 60 segundos. Nessa análise, busca-se manter a número de falsos negativos o mais baixo possível, para que não hajam ataques não detectados.

Figura 5.3: Falsos negativos com relação à variação do valor de *timestamp threshold*, para o mecanismo localizado próximo ao servidor NTP.



Em um cenário como esse, a variação do valor de *timestamp threshold* pode afetar o número de falsos negativos reportados, pois existem ataques com maior intensidade que outros, de forma que utilizar um valor pequeno para esse parâmetro causa a detecção dos ataques mais intensos e a não detecção dos menos intensos. Com base nessa análise, a Figura 5.3 apresenta a porcentagem de falsos negativos com relação à variação do intervalo de tempo utilizado, fixando o valor de *bytes threshold* em 30.000. Pode-se notar a tendência de diminuição no número de falsos negativos com o aumento do *timestamp threshold*. Na primeira medição, em que o *timestamp threshold* = 0.5 segundos, nenhum dos cinco ataques foi detectado e a porcentagem de falsos negativos foi 100%. Contudo, nas medições seguintes pode-se notar que esse número diminuiu até atingir o valor de 20%, no caso em que o *timestamp threshold* assumiu o valor de 3.5 segundos. Nesse

experimento não foi atingido o valor de 100% de verdadeiros positivos, pois o ataque de menor intensidade (6 kbps) nunca foi detectado. Isso se deve ao fato de que esse ataque possui uma média de apenas uma requisição a cada seis segundos, e para que ele fosse detectado, seria necessário um *timestamp threshold* de seis segundos ou mais.

Por fim, os resultados obtidos sugerem que a eficácia do mecanismo, com relação a falsos negativos, é influenciada pelo valor utilizado para o intervalo de tempo máximo entre respostas (*timestamp threshold*). De maneira geral, o valor desse parâmetro deve ser escolhido com cautela, pois da mesma forma que um intervalo grande gera um baixo número de falsos negativos, ele também resulta em uma alta quantia de falsos positivos, conforme a Seção 5.2.1. Além disso, ataques que necessitam de um alto valor para *timestamp threshold* são ataques de baixa intensidade e por isso possuem uma baixa "efetividade". Dessa forma, argumenta-se que pode não ser vantajoso configurar um alto intervalo de tempo para detectar ataques pouco eficientes.

Mecanismo próximo à vítima. Os dois mecanismos se diferenciam, principalmente, pelos contadores de *bytes* ou mensagens, sendo a parte que avalia o intervalo de tempo entre respostas muito similar. Por isso, a avaliação do impacto na mudança de *timestamp threshold* para os mesmos ataques seria equivalente em ambos. Assim, optou-se por omitir a avaliação de falsos negativos para o mecanismo próximo à vítima.

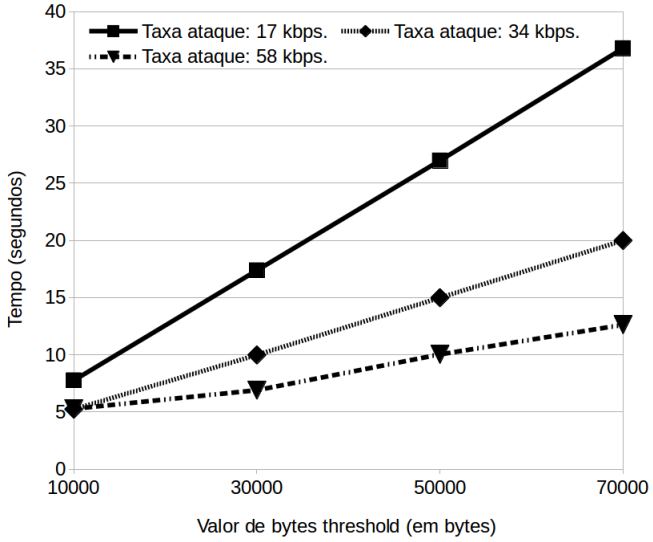
5.2.3 Tempo Decorrido até a Detecção de um Ataque

Mecanismo próximo ao serviço comprometido. Para essa avaliação, utilizou-se um cenário em que há somente um ataque acontecendo, e não há requisições legítimas ocorrendo ao mesmo tempo. O *timestamp threshold* está fixado em três segundos.

A Figura 5.4 ilustra os resultados obtidos na análise do tempo necessário para a detecção de um ataque, de forma que ela representa essa métrica em função do valor empregado para o parâmetro *bytes threshold*. Esse resultado foi mensurado para ataques de intensidades diferentes, ou seja, aumentado a taxa de bits por segundo das respostas NTP `monlist`. Vale ressaltar que as taxas identificadas referem-se apenas a dados do protocolo NTP, excluindo cabeçalhos de outras camadas como Ethernet, IP e UDP.

Pode-se observar que há uma tendência de aumento do tempo necessário para a detecção, visto que quanto maior o valor limite de *bytes*, mais tempo o ataque leva para atingir esse limite. Pode-se notar também que quanto maior for a intensidade do ataque,

Figura 5.4: Mecanismo próximo ao servidor: tempo decorrido até a primeira detecção de um ataque para diferentes taxas de ataques, com relação à variação de *bytes threshold*.



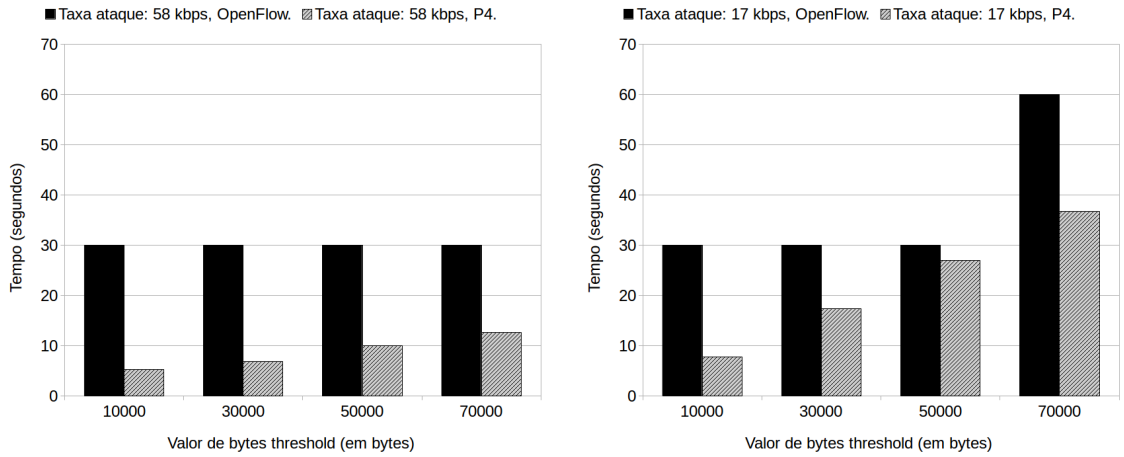
em menos tempo o ataque é reportado. Esse comportamento ocorre, pois quanto mais respostas por segundo recebidas, mais rápido a diferença entre os contadores de *bytes* irá atingir o valor limite.

Por fim, desenvolve-se um exercício com o intuito de avaliar se esse mecanismo, implementado em P4, é comparativamente melhor ou pior do que uma solução baseada em OpenFlow. Considera-se, então, uma situação hipotética em que se tem o mesmo mecanismo proposto neste trabalho, porém implementado em OpenFlow. Ademais, assume-se que o controlador da rede requisite pelas estatísticas do *switch* a cada trinta segundos. O controlador recebe tais informações, faz o devido processamento e identifica se ocorreu (ou não) um ataque.

Com relação à utilização de recursos da rede gerada por esses dois sistemas, argumenta-se que ela seria muito menor em P4, pois o volume de dados carregados em uma mensagem com o intuito de avisar o controlador de que um ataque está ocorrendo seria mínimo, enquanto que o volume transportado pela abordagem OpenFlow seria muito maior. Por exemplo, no caso de uma topologia composta por dezenas de dispositivos, em que cada um deles armazena uma grande quantidade de dados, a carga gerada nessa rede seria consideravelmente alta.

Adicionalmente, a Figura 5.5 apresenta uma possível comparação entre as duas abordagens, com relação ao tempo necessário para a detecção de um ataque enquanto se altera o valor utilizado para *bytes threshold*. Em ambos gráficos, a barra mais escura representa a expectativa para o sistema baseado em OpenFlow, enquanto a barra hachurada

Figura 5.5: Expectativa de comparação do tempo de detecção entre o mecanismo proposto e uma possível estratégia baseada em OpenFlow cujo intervalo de coleta de estatísticas é 30 segundos.



(a) Comparação para um ataque de 58 kbps.

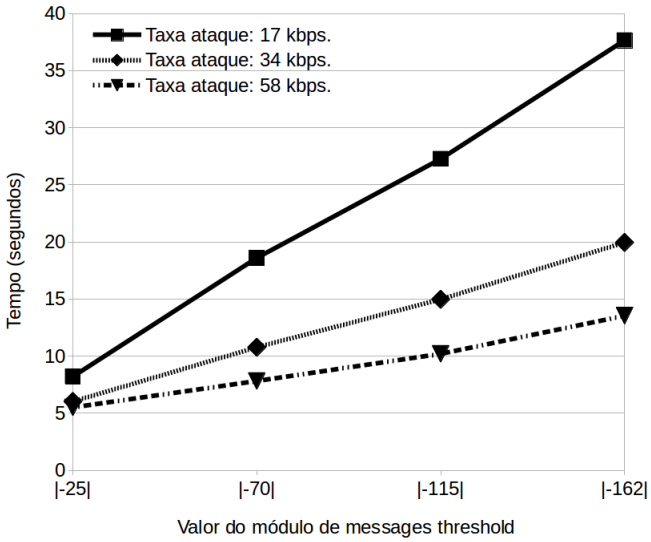
(b) Comparação para um ataque de 17 kbps.

mais clara demonstra o tempo para a detecção via implementação em P4, refletindo os resultados da Figura 5.4. Na Figura 5.5a, ilustra-se um cenário em que o fator de amplificação do servidor NTP é 540 e está ocorrendo um ataque com intensidade de 58 kbps. Como na abordagem OpenFlow a detecção só é efetivada quando o controlador analisa as informações recebidas, e tais informações são requisitadas duas vezes por minuto, o tempo mínimo para a identificação de um ataque seria de trinta segundos. Analisando a aferição em que foi utilizado 70.000 para o valor de *bytes threshold*, observa-se que a detecção do ataque, que em P4 ocorreu próximo do instante doze, para o OpenFlow ocorreria apenas após decorridos trinta segundos.

Da mesma forma, a Figura 5.5b ilustra o mesmo cenário anterior, porém com a ocorrência de um ataque de menor intensidade: 17 kbps. Avaliando novamente o resultado apresentado quando o valor de *bytes threshold* é 70.000, nota-se novamente que o tempo demandado pela abordagem OpenFlow é maior do que para a abordagem P4. Nesse caso, a detecção do mecanismo OpenFlow seria no instante 60 segundos, enquanto que em P4 ela ocorreu próximo ao instante 36. Essa estimativa de 60 segundos se justifica pelo fato de que, quando o controlador analisou os dados provenientes do *switch* no instante 30, tais dados ainda não continham informações que gerassem uma detecção. Somente quando o controlador processou a segunda rodada de informações do dispositivo, no instante 60, é que foi identificado um ataque.

Mecanismo próximo à vítima. Da mesma forma que para o cenário anterior, foi avaliado o tempo necessário para a detecção de três ataques diferentes. Entretanto, para esse mecanismo, foi diversificado o valor de *messages threshold*, ao invés de *bytes threshold*, como pode ser observado na Figura 5.6. A partir desses dados, busca-se comparar o desempenho entre o mecanismo que atua próximo à vitima com o mecanismo localizado nas imediações do servidor NTP. Para que tal fosse possível, foram utilizados valores de *messages threshold* comparáveis com os valores de *bytes threshold* da Figura 5.4. Por exemplo, em ambos os casos estamos lidando com pacotes de resposta de 432 *bytes*. Dessa forma, o limite de 30.000 *bytes*, na Figura 5.4, corresponde a aproximadamente 70 pacotes, que por sua vez está representado na Figura 5.6 através do valor |-70|. Essa relação foi feita para todas as medições realizadas.

Figura 5.6: Mecanismo próximo à vítima: tempo decorrido até a primeira detecção de um ataque para diferentes taxas de ataques, com relação à variação de *messages threshold*.



A partir dessas relações, observa-se que os mecanismos se assemelham muito com relação ao tempo necessário para a detecção de um ataque. Para o ataque de 58 kbps, o mecanismo próximo à vítima (Figura 5.6) identificou um ataque aos 7.8 segundos quando o *message threshold* era -70. Já o mecanismo próximo ao servidor detectou o ataque, para um limite de 30.000 *bytes*, em 6.9 segundos. Segundo os dados obtidos, em 80% das medições realizadas, o mecanismo próximo ao servidor ficou, em média, um segundo mais rápido. Apesar do primeiro mecanismo ter se provado mais ágil, o segundo mecanismo possui a vantagem de utilizar um registrador (de N posições) a menos, ocasionando assim, uma menor quantidade de memória necessária no dispositivo.

6 CONCLUSÃO

Neste trabalho, realizou-se um estudo sobre a possibilidade da utilização de planos de dados programáveis P4 para realizar detecção de intrusão. Mais especificamente, investigou-se como essa nova tecnologia pode ser empregada para a detecção de ataques de negação de serviço baseados em amplificação NTP. Em seguida, baseado nos conhecimentos adquiridos, propôs-se dois mecanismos para identificar atividades maliciosas dessa natureza. Posteriormente, desenvolveu-se ambientes em P4 capazes de abordar tal problema e, por fim, avaliou-se as estratégias implementadas.

Dentre os objetivos desse trabalho, almejava-se avaliar o comportamento dos mecanismos implementados com relação à variação de alguns parâmetros que ditam o funcionamento "normal" da rede no qual estão inseridos. O primeiro desses parâmetros foi o intervalo de tempo máximo suportado entre duas respostas NTP `monlist` consecutivas, denominado *timestamp threshold*. O segundo parâmetro foi o limite entre requisições e respostas NTP `monlist`, que no primeiro mecanismo se relacionava com a diferença entre *bytes*, e no segundo mecanismo se relacionava com a diferença na quantidade de mensagens.

Com base nos resultados obtidos, constatou-se que o tempo e o limite entre requisições e respostas possuem um papel fundamental na calibragem desses mecanismos. Em ambos mecanismos, foi possível obter de zero a cem por cento de falsos positivos variando apenas esses dois parâmetros. Para situações de ataques, foi constatado que o número de falsos negativos também pode obter uma variação semelhante ao diversificar o valor do intervalo de tempo utilizado. De maneira geral, os mecanismos propostos atingem melhores resultados quando o ataque possui uma "assinatura" muito agressiva. Utilizando as medições feitas, quanto maior a intensidade do ataque, menor seria o *timestamp threshold* necessário para detectá-lo e, em resultado disso, menor seria o número de falsos positivos observados.

Por fim, foi analisado o tempo necessário para que cada um dos mecanismos conseguisse detectar um ataque. Observou-se que, dependendo da intensidade do ataque, o tempo decorrido também sofre variações. De fato, encontrou-se uma tendência em termos de que, quanto mais intenso, menor o tempo necessário para a detecção. Além disso, através de um exercício analítico, foi apresentado que esses mecanismos possuem um tempo de detecção menor do que uma possível estratégia baseada em OpenFlow convencional.

A partir das conclusões evidenciadas aqui, lembra-se que este trabalho não teve

como objetivo criar um mecanismo extremamente robusto e tolerante a diversos tipos de ataques. O seu escopo foi, sim, escolher uma ação maliciosa, cuja assinatura para detecção fosse relativamente simples, e tentar explorar a implementação de detectores no contexto de planos de dados programáveis P4.

Como trabalho futuro, busca-se validar os resultados obtidos utilizando dados reais, ou seja, gerar ataques com alta taxa de dados ou até mesmo utilizar traços de ataques verídicos. Além disso, almeja-se aprimorar os mecanismos de forma que eles sejam tolerantes a atacantes "inteligentes". Isto é, que consigam identificar ações maliciosas que possuam um certo conhecimento sobre os parâmetros de detecção (como por exemplo, os limites de tempo e de *bytes*) e usem isso a seu favor. Uma forma de abordar esse problema seria com o uso de *thresholds* adaptativos, em que o mecanismo implementaria uma forma de aprender o comportamento "normal" da rede, e calibraria seus *thresholds* de acordo com essas medidas. Em um contexto mais amplo, aspira-se que esse trabalho possa ser utilizado como alicerce para detecção de outros tipos de ataques de amplificação, baseados em diferentes protocolos como: DNS, SNMP e SSDP.

REFERÊNCIAS

AFEK, Y.; BREMLER-BARR, A.; SHAFIR, L. Network anti-spoofing with sdn data plane. In: **IEEE INFOCOM 2017 - IEEE Conference on Computer Communications**. [S.l.: s.n.], 2017. p. 1–9.

AKAMAI TECHNOLOGIES INC. **State of the Internet Security Report Q1 2017**. [S.l.], 2017. Available from Internet: <https://www.stateoftheinternet.com/>.

AKAMAI TECHNOLOGIES INC. **State of the Internet/Security Report Q2 2017**. [S.l.], 2017. Available from Internet: <https://www.stateoftheinternet.com/>.

ARBOR NETW. INC. **Arbor Networks Special Report: Worldwide infrastructure security report volume XII**. [S.l.], 2017. Available from Internet: <https://www.arbornetworks.com/insight-into-the-global-threat-landscape>.

BACKES, M. et al. On the feasibility of ttl-based filtering for drdos mitigation. In: **Research in Attacks, Intrusions, and Defenses: 19th International Symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings...** Cham: Springer International Publishing, 2016. p. 303–322. Available from Internet: <https://doi.org/10.1007/978-3-319-45719-2_14>.

BAREFOOT NETWORKS. **Barefoot Tofino**. 2017. Available from Internet: <https://www.barefootnetworks.com/technology>.

BOSSHART, P. et al. P4: Programming protocol-independent packet processors. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 44, n. 3, p. 87–95, jul. 2014. Available from Internet: <http://doi.acm.org/10.1145/2656877.2656890>.

BRAGA, R.; MOTA, E.; PASSITO, A. Lightweight ddos flooding attack detection using nox/openflow. In: **IEEE Local Computer Network Conference**. [S.l.: s.n.], 2010. p. 408–415.

DOULIGERIS, C.; MITROKOTSA, A. Ddos attacks and defense mechanisms: Classification and state-of-the-art. **Comput. Netw.**, Elsevier North-Holland, Inc., New York, NY, USA, v. 44, n. 5, p. 643–666, abr. 2004.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The road to sdn: An intellectual history of programmable networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 44, n. 2, p. 87–98, abr. 2014. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/2602204.2602219>.

GIOTIS, K. et al. Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments. **Comput. Netw.**, Elsevier North-Holland, Inc., New York, NY, USA, v. 62, p. 122–136, abr. 2014. Available from Internet: <http://dx.doi.org/10.1016/j.bjp.2013.10.014>.

KAMBOURAKIS, G. et al. A fair solution to dns amplification attacks. In: **Second International Workshop on Digital Forensics and Incident Analysis (WDFIA 2007)**. [S.l.: s.n.], 2007. p. 38–47.

KREIBICH, C. et al. Using packet symmetry to curtail malicious traffic. **ACM Hotnets-IV**, v. 200, 2005.

KÜHRER, M. et al. Exit from hell? reducing the impact of amplification ddos attacks. In: **23rd USENIX Security Symposium (USENIX Security 14)**. San Diego, CA: USENIX Association, 2014. p. 111–125.

LI, Y. et al. Flowradar: A better netflow for data centers. In: **13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)**. Santa Clara, CA: USENIX Association, 2016. p. 311–324. Available from Internet: <https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/li-yuliang>.

LIU, Z. et al. One sketch to rule them all: Rethinking network flow monitoring with univmon. In: **Proceedings of the 2016 ACM SIGCOMM Conference**. New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 101–114. Available from Internet: <http://doi.acm.org/10.1145/2934872.2934906>.

P4 LANGUAGE CONSORTIUM. **Behavioral Model (BMv2)**. 2017. Available from Internet: <https://p4.org/code/>.

PAXSON, V. An analysis of using reflectors for distributed denial-of-service attacks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 31, n. 3, p. 38–47, jul. 2001. Available from Internet: <http://doi.acm.org/10.1145/505659.505664>.

POPESCU, D. A.; ANTICHI, G.; MOORE, A. W. Enabling fast hierarchical heavy hitter detection using programmable data planes. In: **Proceedings of the Symposium on SDN Research**. New York, NY, USA: ACM, 2017. (SOSR '17), p. 191–192. Available from Internet: <http://doi.acm.org/10.1145/3050220.3060606>.

PRINCE, M. **Technical Details Behind a 400Gbps NTP Amplification DDoS Attack**. Cloudflare, Inc, 2014. Available from Internet: <https://blog.cloudflare.com/technical-details-behind-a-400gbps-ntp-amplification-ddos-attack/>.

ROSSOW, C. Amplification hell: Revisiting network protocols for ddos abuse. In: **NDSS**. [S.l.: s.n.], 2014.

SHIN, S. et al. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In: **Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security**. New York, NY, USA: ACM, 2013. (CCS '13), p. 413–424. Available from Internet: <http://doi.acm.org/10.1145/2508859.2516684>.

SIVARAMAN, V. et al. Heavy-hitter detection entirely in the data plane. In: **Proceedings of the Symposium on SDN Research**. New York, NY, USA: ACM, 2017. (SOSR '17), p. 164–176. Available from Internet: <http://doi.acm.org/10.1145/3050220.3063772>.

TEAM, M. **Mininet: An instant virtual network on your laptop (or other PC)**. 2012.

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**, v. 18, n. 1, p. 602–622, Firstquarter 2016.

YAO, G.; BI, J.; XIAO, P. Source address validation solution with openflow/nox architecture. In: **2011 19th IEEE International Conference on Network Protocols**. [S.l.: s.n.], 2011. p. 7–12.

ZAALOUK, A. et al. Orchsec: An orchestrator-based architecture for enhancing network-security using network monitoring and sdn control functions. In: **2014 IEEE Network Operations and Management Symposium (NOMS)**. [S.l.: s.n.], 2014. p. 1–9.

ZARGAR, S. T.; JOSHI, J.; TIPPER, D. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. **IEEE Communications Surveys Tutorials**, v. 15, n. 4, p. 2046–2069, Fourth 2013.

APÊNDICE A — TRABALHO DE GRADUAÇÃO I

Detecção de Ataques em Redes Utilizando P4

Roberto Oppermann Cordoni[1], Luciano Paschoal Gaspary[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Av. Bento Gonçalves, 9500 – Porto Alegre – RS – Brazil

{rocordoni, paschoal}@inf.ufrgs.br

Resumo. *Esse trabalho apresenta um levantamento sobre metodologias e técnicas de detecção de ataques baseadas em análise de fluxo, que em conjunto com a ideia de planos de dados programáveis, podem ser usadas para detecção de diferentes tipos de ataques de segurança. Para o Trabalho de Graduação 2, propõe-se a implementação de diferentes técnicas na linguagem P4, além da prototipação de um ambiente de experimentação onde será possível a análise da eficiência de tais técnicas.*

1. Introdução

Aliada com a rápida expansão da internet ocorrida nas últimas décadas, a quantidade de ataques cibernéticos reportados também apresentou um alto crescimento. O prejuízo associado a tais ataques não fica limitado à esfera financeira, como a perda de milhares de dólares caso um serviço fique fora do ar, mas também pode causar despesas no sentido da perda de clientes/reputação de uma certa empresa. Considerando essa situação, é importante que a detecção de tais ameaças seja extremamente rápida, para que ações cabíveis sejam tomadas antes que o dano seja maior. Essa tarefa, porém, é desafiadora devido à diversidade de ataques existentes (varreduras, vírus, *worms*, DDoS, etc.).

Atualmente, técnicas de amostragem de pacotes baseadas em fluxo são amplamente utilizadas na indústria. O problema associado a essas metodologias é o custo computacional necessário para o processamento e classificação desses fluxos quando se trabalha com taxas de pacotes extremamente altas. Endereçando esse problema, a indústria vem apresentando recentemente uma nova geração de dispositivos de encaminhamento que podem operar a taxas de até 100 Gbps, além de também poderem ser programados para executar diversos comandos sobre cada pacote, utilizando uma linguagem denominada P4.

Apesar dos diversos avanços na área de segurança em redes de computadores, (que reúne soluções clássicas de análise de fluxos), as abordagens existentes desconsideram o uso de planos de dados programáveis para a detecção de ataques. Para o Trabalho de Graduação 2, propõe-se o estudo e a prototipação de programas de detecção de ataques implementados em P4. Esses programas serão projetados com base em técnicas de análise de fluxo, como será descrito na Seção 3. Além disso, como forma de entender como os mecanismos a serem implementados são eficientes, apresenta-se um ambiente de experimentação para esses mecanismos, cujo resultado indicará como variações de diferentes parâmetros podem influenciar no desempenho

das detecções.

O restante desse artigo está organizado como segue. Na Seção 2 e 3 apresenta-se uma análise do estado da arte de trabalhos relacionados à detecção de ataques, mais especificamente utilizando a abordagem por fluxos. Na Seção 4 são descritos os planos de dados programáveis, além de nova linguagem criada para a programação de dispositivos de encaminhamento, chamada P4. A Seção 5 apresenta um delineamento sobre o trabalho. A Seção 6 define a proposta deste trabalho. Na Seção 7 descreve-se a metodologia a ser utilizada no desenvolvimento do trabalho. Na Seção 8 apresenta-se o cronograma de atividades previsto para o andamento da pesquisa. Por fim, na Seção 9 são apresentadas as considerações finais.

2. Detecção de Ataques

Nesta seção, são apresentadas diferentes classificações e taxonomias comumente utilizadas na área de segurança. A seção começa caracterizando os diferentes tipos de ataques existentes, e termina discorrendo sobre as diferentes terminologias utilizadas para identificar os mecanismos de detecção atuais.

2.1. Classificação de Ataques

Diversas classificações de ataques de rede foram descritas na literatura. Pode-se distribuir essas classificações nas seguintes definições (Sperotto *et al.*, 2010):

- **Ataques físicos**: Ataques com o intuito de danificar o dispositivo de rede e/ou o sistema.
- **Buffer overflow**: Ataques que tentam controlar ou causar falha no sistema de destino por meio de um estouro de *buffer*.
- **Ataques de palavra-chave**: Ataques que tentam obter a chave de acesso para sistemas protegidos.
- **Ataques de negação de serviço (distribuído ou não)**: Ataques que resultam em usuários legítimos tendo dificuldade de acesso a determinado recurso, ou até mesmo recusa total.
- **Ataques de coleta de informações**: Esse tipo de ataque não compromete diretamente o sistema de destino, porém obtém informações sobre o mesmo para um possível ataque futuro. Exemplo de ataques desse tipo são *traffic sniffing* e *port scans* (varredura de portas).
- **Cavalo de Troia**: Um *software* malicioso disfarçado de uma aplicação benigna que executa ações indesejadas.
- ***Worms***: Programas que se autorreplicam sem o uso de arquivos infectados. Normalmente *worms* se propagam pelos serviços de rede em um computador ou através de e-mail.
- **Vírus**: Programas que se autorreplicam, porém com o uso de arquivos infectados. Vírus necessitam a interação com o usuário para se propagar.

É possível ainda adicionar *botnets* às categorias especificadas anteriormente**.** *Botnets* são um conjunto de computadores afetados por um programa malicioso, remotamente controlados por um programa mestre, que operam contra as intenções do usuário e sem o seu conhecimento. *Botnets* tendem a ser um ambiente ótimo para ataques de natureza distribuída, como DDoS ou SPAMs.

2.2. Mecanismos de Detecção

Estudos sobre detecção de intrusão aparecem na literatura desde a década de 1980. Desde então, diversas taxonomias foram propostas para as diferentes técnicas existentes. Entre elas se destaca o trabalho de Debar *et al. (*Debar *et al., 1999)*, que foi um dos pioneiros a elaborar uma taxonomia para o tema. A sua pesquisa é norteada nos seguintes aspectos:

- **Método de detecção**: Um sistema de detecção é dito baseado em comportamento *(behavioral-based)* se ele segue algum tipo de comportamento dito "normal", onde qualquer desvio desse padrão pode ser considerado um ataque. Já se o sistema usa uma combinação dos dados de entrada com a definição de um determinado ataque, ele é considerado baseado em conhecimento *(knowledge-based).* Na literatura, sistemas desses tipos são usualmente referidos como *anomaly-based* e *misused-based*, respectivamente.
- **Resposta a ataques**: Um sistema pode ser dito passivo se, ao detectar um comportamento estranho, ele apenas gera um alerta que será processado posteriormente por um outro sistema ou por uma pessoa. Ou o sistema pode ser considerado ativo se ele pró-ativamente executa alguma ação contra o ataque.
- **Fontes de análise**: A análise feita a partir da monitoração das funcionalidades internas de uma única fonte (uso de CPU, arquivos de *log* do sistema) é dita *host-based.* Já se a análise é feita com base em informações globais sobre a rede, o método é dito *network-based.*
- **Frequência de análise**: O sistema pode executar a análise dos dados periodicamente, ou logo que os dados estejam prontos *(continuous monitoring).*

Axelsson (Axelsson, 2000) estendeu a taxonomia de Debar de maneira que além das características citadas acima, um sistema também pode ser classificado com base nos seguintes aspectos:

- **Granularidade dos dados:** Essa categoria se refere a sistemas que processam os dados coletados de duas formas: continuamente ou em lotes (*batches)*.
- **Localização do processamento dos dados**: O sistema pode ser centralizado ou distribuído em relação à origem dos dados.
- **Localização da coleta dos dados**: A coleta de dados pode ser centralizada ou distribuída.
- **Nível de interoperabilidade**: O sistema pode trabalhar trocando dados em conjunto com outros sistemas, ou pode trabalhar isoladamente.

Como ficará mais claro na sequência do artigo, no escopo deste trabalho serão enfatizados processos de detecção de ataques baseados em fluxo. Por essa razão, a próxima seção aprofunda esse conceito.

3. Detecção Baseada em Fluxo

Essa seção apresenta técnicas para detecção de ataques focadas na análise por fluxo. Na primeira parte será discorrido o ataque de negação de serviço. Depois, a discussão trata de *worms* e por fim, *botnets.*

3.1. Negação de Serviço

Pode-se citar dois tipos de ataques de negação de serviço: os ataques do tipo força bruta, que são baseados na sobrecarga ou exaustão de um certo recurso no sistema de destino, e os ataques baseados em semântica, os quais fundamentam-se em causar a interrupção de um determinado serviço pelo envio de uma mensagem mal construída que pode gerar falha no computador de destino. Os ataques deste último tipo, como se pode imaginar, são extremamente difíceis de serem diretamente detectados porque eles não causam nenhuma disformidade na frequência ou intensidade do fluxo. Já os ataques de força bruta são mais facilmente identificados pois podem causar perceptíveis variações no volume do tráfego para certos fluxos.

Diversos trabalhos foram propostos na temática de detecção de anomalias em redes de alta velocidade que podem significar ataques de negação de serviço. Primeiramente, Gao *et al.* (Gao *et al.*, 2006) abordaram o problema usando agregações de fluxos, também chamadas de *sketches*. Um *sketch* é originalmente uma tabela *hash* unidimensional para rápido armazenamento de informações, cujo principal objetivo é computar o número de ocorrências de um certo evento. Pode-se citar um exemplo do trabalho de *Gao,* onde o autor propõe o uso de *sketches* para a detecção de um ataque de negação de serviço do tipo *SYN Flooding.* Para esse caso o *sketch* armazena a diferença entre o número de pacotes *SYN* e o número de *SYN/ACKs* para cada tupla (Dest IP, Dest Port) em um determinado intervalo de tempo. Caso o valor resultante seja distinto, o algoritmo trata esse fluxo como um potencial ataque.

Outra abordagem a ser considerada é a apresentada por Kim *et al. (*Kim *et al., 2004)*. Em seu trabalho, os autores identificam diversos tipos de ataques de negação de serviço, todos descritos em termos de padrões de tráfego. Mais especificadamente, o autor concentra-se em: número de fluxos, número de pacotes, tamanho do fluxo, tamanho de pacotes e largura de banda utilizada, como exemplificado na Figura 1. Novamente pode-se citar o exemplo de um ataque do tipo *SYN Flooding*, que possui um padrão do tipo: alto número de fluxos, baixo número de pacotes e baixo tamanho de fluxo, além de pacotes com poucos bytes. Esse padrão é relativamente diferente do padrão gerado por ataques de inundação ICMP ou UDP, os quais possuem grande consumo de largura de banda e alto número de pacotes transferidos. Essas classificações são formalizadas pelo autor em *funções de detecção,* as quais são usadas para obter a probabilidade de que um certo fluxo represente um possível ataque.

Por último, é importante citar o estudo de Munz *et al. (*Munz *&* Carle*, 2007)*,

que propôs o TOPAS (*Traffic flOw and Packet Analysis System)*, um sistema coletor de informações de fluxos a partir de muitas fontes em diferentes localidades, com o intuito de oferecer esses dados de forma adequada para um pós-processamento. Com essa plataforma, diversos módulos podem ser executados concorrentemente de acordo com as necessidades do operador. Pode-se citar alguns módulos descritos, como: *SYN Flood Detection Module, Traceback Module* (identificação do ponto de entrada de pacotes maliciosos na rede) e *Web Server Overloading Module* (detecção de ataques DoS usando requisições HTTP).

3.2. Worms

O ataque do tipo *worm* pode ser dividido em duas fases: a fase da descoberta do alvo, onde o *worm* busca na rede um sistema vulnerável que possa ser atacado, e a fase de transferência de código. Em abordagens baseadas em fluxo, a segunda fase do ataque (transferência) não pode ser utilizada para detecção da ameaça visto que seria necessária uma análise do *payload* do pacote, e não somente de seus cabeçalhos. Por isso, detecções baseadas em fluxo se concentram na fase de descoberta do alvo e não na de transferência.

Os trabalhos de Dübendorfer *et al.* (Dübendorfer & Plattner, 2005) e Wagner *et al. (*Wagner *et al., 2005)* tentaram classificar o comportamento do atacante com base em conexões de entrada e saída, agrupando-os em três classes diferentes. A classe de tráfego engloba sistemas que enviam mais tráfego do que recebem; a classe de conectores, a qual é caracterizada por ameaças que exibem número não usual de conexões de saída; e finalmente a classe de replicadores que agrupa hospedeiros envolvidos em muitas conexões bidirecionais. O método busca detectar a propagação de *worms* ao periodicamente verificar o estado dos nós da rede, visto que propagações causam mudanças notáveis na métrica de uma ou mais classes.

3.3. Botnets

Como mencionado anteriormente, *botnets* são um conjunto de máquinas infectadas que permitem ao atacante o controle remoto de seus recursos computacionais para executar atividades maliciosas. Esse conjunto de máquinas compartilha um software em comum, o qual as liga com uma infraestrutura de Comando e Controle (C&C). Isto permite que *botnets* tenham duas arquiteturas básicas: centralizada ou distribuída.

No modelo centralizado, a comunicação entre o *botmaster* e os clientes da *botnet* se dá através de uma infraestrutura centralizada. Podem ser usados diversos tipos de protocolos na comunicação entre essas duas entidades, mas os mais comuns são *HTTP*, *DNS* e *IRC*. Já para arquiteturas descentralizadas, as informações sobre a rede *botnet* estão espalhadas ao longo de toda a rede fraudulenta. Dessa forma, esse tipo de infraestrutura é mais difícil de ser explorado, visto que mesmo a descoberta de diversos *bots* não acarreta a perda da *botnet* inteira.

Attacks / Property	scanning		flooding				
	host	network	TCP SYN	smurf	fraggle	ping-pong	general (ICMP,UDP,TCP) flooding
flow count	L	L	L	L	L	S	L or S
flow size/flow	S	S	S	L or S	L or S	L	L or S
packet count/flow	S	S	S	L or S	L or S	L	L or S
packet size	S	S	S	L or S	L or S	L or S	L or S
total bandwidth	L or S	L or S	L or S	L	L	L	L
total packet count	L or S	L or S	L or S	L	L	L	L
property	1 destination	1 port		ICMP broadcast	UDP broadcast	reflecting port	

Figura 1. Caracterização de padrões de ataques.

Um trabalho a se destacar é o de Haddadi *et al.* (Haddadi *et al.*, 2014), onde buscou-se a detecção de *botnets* através da análise de fluxo para tráfegos *HTTP*. Em suma, os autores identificaram mais de 20 características para cada fluxo observado. Entre elas, destacam-se a duração do fluxo, o tamanho do fluxo em bytes e o tipo de serviço. A partir desses indicativos e de algoritmos classificadores, os autores alcançaram uma taxa de precisão de 88%.

Por fim, a Figura 1 sumariza alguns padrões de fluxo para diferentes tipos de ataques, onde L significa “alta quantidade” e S significa “baixa quantidade”. Essa figura foi retirada do trabalho de Kim *et al.* (Kim *et al*., 2004).

Após terem sido apresentados mecanismos de detecção de ataques, com ênfase naqueles baseados em fluxo, apresenta-se a seguir um novo paradigma de processamento nos dispositivos de encaminhamento conhecido como plano de dados programável.

4. Programação do Plano de Dados com P4

A programabilidade da rede é um assunto que vem sendo abordado há mais de vinte anos pela comunidade de pesquisa. Desde então, redes definidas por software (SDN) têm causado uma revolução na maneira com que se projeta e se gerencia redes. Como uma das principais características de SDNs, pode-se citar o desacoplamento entre o plano de controle (que decide como manipular o tráfego) e o plano de dados (cuja função é o encaminhamento de tráfego baseado nas decisões do plano de controle).

O plano de controle pode ser visto como um software que gerencia múltiplos dispositivos do plano de dados (roteadores, *switches*, etc), utilizando APIs definidas para esse propósito, como o OpenFlow. Um dispositivo com suporte a OpenFlow pode se comportar como um roteador, *switch*, *firewall*, etc dependendo das regras que lhe foram instaladas pelo controlador de rede. Em sua primeira especificação, o protocolo OpenFlow contava com pouquíssimas tabelas e campos de cabeçalho. Porém, com o intuito de aprimorar a comunicação entre os dispositivos de rede e o controlador, o protocolo foi sendo desenvolvido e muitos outros tipos de cabeçalhos foram sendo adicionados à especificação, totalizando mais de 40 tipos diferentes até o final de 2013.

Bosshart *et al.* (Bosshart *et al*., 2014) declararam que em vez de continuamente aumentar a especificação do OpenFlow, os dispositivos de rede deveriam suportar mecanismos de análise de pacotes mais flexíveis, onde o plano de controle definiria tal analisador utilizando uma nova interface de comunicação com o dispositivo. Tal interface deveria ser simples, elegante e mais difícil de se tornar obsoleta do que a atual

especificação do OpenFlow. Dessa forma, os autores propuseram uma linguagem de alto nível para programar processadores de pacotes independentes de protocolo, denominada P4 (do inglês, *Programming Protocol-independent Packet Processors*).

Um dos principais requisitos para aplicações ligadas ao plano de dados, onde P4 foi planejado para atuar, é a capacidade de processar pacotes a uma taxa extremamente alta (tipicamente entre 10 e 100 Gbps, em dezenas de portas simultaneamente). Para atingir tais taxas, algoritmos de encaminhamento de pacotes eram usualmente implementados usando hardware dedicado. Contudo, o projeto de hardware é uma atividade complexa e de difícil reconfiguração em campo. Assim, para testar e lançar um novo protocolo, ou até mesmo para modificar um sistema existente, pesquisadores eram obrigados a se dedicar em novos projeto de hardware (uma atividade demorada e onerosa). Dessa forma, a comunidade científica tentou durante muitos anos construir roteadores programáveis para que fosse possível o desenvolvimento de novas aplicações para dispositivos já finalizados.

De acordo com Sivaraman *et al.* (Sivaraman *et al*., 2016), diversas pesquisas na área de roteadores programáveis resultaram em equipamentos cujo desempenho era diversas ordens de grandeza abaixo do necessário para tais dispositivos, e por isso eles nunca foram realmente utilizados em produção. Todavia, recentemente novos tipos de chips ASICs programáveis foram sendo disponibilizado pela indústria. Esses novos chips são capazes de competir, em termos de desempenho, com chips de funções fixadas (*fixed-functions*) distribuídos atualmente. Em termos de programabilidade, esses novos chips permitem o operador de rede especificar como analisar/encaminhar pacotes sem restrições em relação ao formato de protocolo ou ao conjunto de ações que podem ser executadas quando há correspondência de pacotes em uma tabela do tipo *match-action.*

A linguagem P4 pode então usar esse novo tipo de chip para configurar um dispositivo de encaminhamento, determinando de maneira flexível como pacotes devem ser processados e ao mesmo tempo garantindo alta taxa de dados. Assim, o programador não fica preso a um certo conjunto fixado de instruções oferecido por um equipamento não programável. De maneira geral, a Figura 2 mostra a relação entre P4 e APIs existentes (como OpenFlow) utilizadas para popular as tabelas de encaminhamento do equipamento programável. Os três principais objetivos da linguagem são:

- **Reconfigurabilidade**: O controlador deve possuir a capacidade de configurar e modificar as capacidades do dispositivo.
- **Independência de protocolo**: O equipamento não deve ser limitado em termos de protocolos suportados. Em vez disso, o controlador deve ser capaz de especificar: o *parser* para os possíveis tipos de formatos de cabeçalho e também ações a serem executadas sobre esses *headers.*

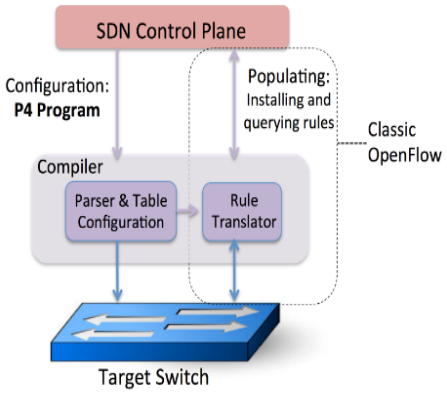


Figura 2. Linguagem P4 para configuração de *switches*.

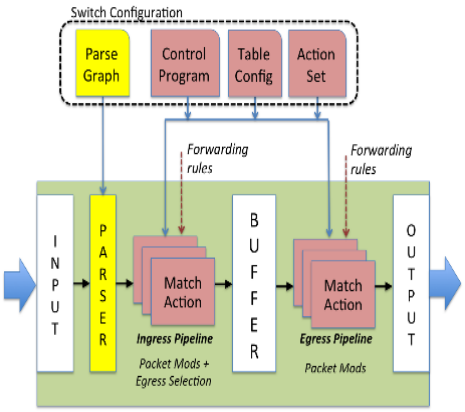


Figura 3. Modelo para *switches* programáveis.

- **Independência de hardware**: Não deve ser necessário para o controlador ter conhecimento do hardware do dispositivo a ser configurado. Isso deve ser trabalho do compilador.

4.1. P4: Modelo para Processadores de Pacotes

O modelo abstrato construído pelos idealizadores do P4 define, de uma forma genérica, como pacotes devem ser processados em diferentes dispositivos de encaminhamento (roteadores, *switches)* e tecnologias (*fixed-functions* ASICs, NPUs, FPGAs). Assim, é possível a definição de uma linguagem para representar o comportamento de tais processadores de pacotes em termos desse modelo. Dessa forma, programadores podem criar programas independentes da plataforma de destino e o compilador se encarregará de mapeá-lo para diferentes tipos de dispositivos.

Esse modelo define que *switches* devem encaminhar dados por meio de um analisador de pacotes programável, seguido por múltiplos estágios de tabelas do tipo *match-action* (organizadas em série, paralelo ou uma combinação de ambos), como exemplificado na Figura 3. De acordo com essa figura, pacotes ingressantes são primeiramente direcionados para o analisador (*parser*). Ele, então, extrai os campos do cabeçalho do pacote e os envia para as tabelas de *match-action.* Pode-se dizer que por definir os tipos de campos a serem extraídos, o analisador caracteriza o programa e determina os tipos de protocolos que o dispositivo suporta.

As tabelas de *match-action* são divididas em *ingress* e *egress.* As tabelas de ingresso, além de determinar a porta de saída e a fila de destino do pacote, também estabelece o que fazer com ele. Algumas alternativas possíveis são: encaminhamento, replicação, ignorar, etc. Já as tabelas de egresso podem ser utilizadas para modificar cabeçalhos de pacotes, se necessário. Além disso, é possível ainda utilizar *action-tables* para manter informações sobre determinado pacote/fluxo, como contadores.

Ademais, a linguagem possibilita que pacotes carreguem consigo informações adicionais por entre os estágios do *pipeline*. Esse tipo de informação é chamada de metadado (*metadata)*. Esses metadados podem ser, por exemplo: o identificador da

porta de ingresso, o *timestamp* do pacote ou até mesmo a fila de ingresso/egresso.

4.2. Conceitos de P4

Um programa P4 possui cinco componentes básicos:

- **Cabeçalhos (h*eaders)***: a construção do programa começa com a especificação dos formatos de cabeçalhos válidos. Para cada cabeçalho deve-se definir uma lista ordenada contendo os diferentes campos acompanhados de seus respectivos tamanhos.
- **Analisador *(parser)***: P4 implementa a análise de cabeçalhos como uma máquina de estados. Para isso, ele assume que o dispositivo de destino é capaz de implementar uma máquina capaz de analisar o *header* de cada pacote do início ao fim, extraindo seus diversos campos um por um. O processo de análise do cabeçalho começa no estado inicial (*start)* e prossegue até que o estado final (*stop*) seja atingido, ou até que uma transição não definida seja encontrada (caracterizando um erro).
- **Tabelas**: as tabelas declaram como os valores obtidos previamente dos cabeçalhos devem ser usados e também quais ações devem ser tomadas caso haja correspondência em alguma tabela.
- **Ações**: a própria linguagem P4 já estabelece uma série de operações primitivas. Cada programa, por sua vez, pode agrupar um conjunto dessas primitivas e definir uma ação. Alguns exemplos de primitivas são: ***remove_header***, que retira certo campo do cabeçalho de um pacote e ***increment***, que incrementa o valor de um campo.
- **Programa de controle**: depois de definido o *parser*, as tabelas e as ações do programa, o programa de controle é responsável por especificar o fluxo a ser seguido entre uma tabela e outra.

Esses componentes juntos definem um programa escrito na linguagem P4. O compilador, então, será responsável por analisar esse programa e gerar a configuração final para o dispositivo de destino.

Em resumo, P4 mostra-se como uma alternativa para os *switches* de funções definidas, capazes de reconhecer apenas um número finito de cabeçalhos e que processam pacotes utilizando um conjunto predeterminado de ações. A linguagem propõe um avanço para o modo como dispositivos pertencentes ao plano de dados são configurados (e modificados) em campo. Dessa forma, um programador deve estabelecer o comportamento de um equipamento, sem conhecer detalhes sobre a arquitetura de destino. Logo, a tarefa de transformar o programa de controle para código que pode ser mapeado para diferentes dispositivos será executada pelo compilador.

4.3. Detecção de Anomalias de Rede Usando P4

Atividades de pesquisa recentes na área mostram que é possível abordar problemas de segurança em redes utilizando essa nova linguagem. Por exemplo, Li *et al.* (Li *et al*.,

2016) propuseram uma ferramenta de monitoração baseada no NetFlow, chamada FlowRadar. Esse mecanismo, implementado em P4, consegue armazenar contadores para todos os fluxos existentes com pouca utilização de memória, usando *encoded flowsets*. Além disso, ele é capaz de exportar as informações armazenadas em intervalos de 10 ms. A principal vantagem do FlowRadar é a capacidade de identificar a melhor divisão de trabalho entre *switches* com baixo poder de processamento e controladores com recursos computacionais relativamente superiores. Esse método é interessante na detecção de ataques de rede devido ao fato de que se pode reportar um ataque ao controlador rapidamente, visto que estatísticas são enviadas ao coletor em intervalos relativamente curtos.

Além disso, Sivaraman *et al.* (Sivaraman *et al*., 2017) sugeriram um algoritmo que armazena estatísticas para os *k* fluxos mais pesados (*Heavy Hitters)*, delimitado pelas restrições de uso de memória de *switches* programáveis. O algoritmo, chamado *HashPipe*, atinge seu objetivo identificando menos de 5% de falsos negativos e 0.001% de falsos positivos ao lidar com um total de 300 *heavy hitters,* utilizando apenas 80KB de memória. A identificação de fluxos pesados na rede pode ser vista como uma forma de abordar os problemas de detecção de ataques e também verificar anomalias de rede. Estendendo o trabalho de Sivaraman, Popescu *et al.* (Popescu *et al.*, 2017) utilizaram a programabilidade do plano de dados para transformar o *switch* em um dispositivo "ativo", e não mais passivo. No algoritmo proposto, o controlador não participa inteiramente na atividade de monitoração, ou seja, o plano de dados somente entrará em contato com o plano de controle quando a detecção de uma atividade maliciosa for efetivada.

Portanto, nota-se que é possível usar o plano de dados para a construção de novas metodologias de análise de diversos aspectos relacionados à segurança em redes de computadores. Além disso, por se tratar de um assunto relativamente novo, ainda existem inúmeras possibilidades passíveis de serem exploradas, principalmente na área de detecção de ataques.

5. Delineamento do Trabalho

Como mencionado por Sivaraman *et al.* (Sivaraman *et al.*, 2017), identificar fluxos de alto volume diretamente no plano de dados é importante para diversas aplicações, como a detecção de ataques e anomalias, aplicações para engenharia de tráfego e também mecanismos de roteamento baseados em tamanho de fluxo (*flow-size aware routing)*.

Abordagens baseadas em análise de fluxo e técnicas de amostragens de pacotes na forma de NetFlow são largamente utilizadas em dispositivos de encaminhamento atuais. Porém, o custo associado ao processamento de amostragem de pacotes em software torna inviável a amostragem de pacotes a taxas relativamente altas (comumente só um a cada mil pacotes são examinados na prática). Na tentativa de identificar ataques/anomalias, alguns métodos de intrusão buscaram analisar o conteúdo (*payload)* de cada pacote. O problema com essa abordagem é que se torna extremamente difícil manipular o conteúdo de cada pacote quando se opera a uma taxa de múltiplos Gbps.

A nova geração de *switches* programáveis soluciona essa necessidade de uma alta taxa de processamento. Eles conseguem operar em taxas de até 100 Gbps e ainda podem ser programados para armazenar dados sobre diversos pacotes/fluxos. Por esses motivos, o método escolhido para ser trabalhado nesse projeto foi a implementação de algoritmos de detecção baseados em fluxo na linguagem para dispositivos programáveis P4. Como mencionado anteriormente, com essa escolha é possível identificar ataques observando os diferentes padrões de fluxos de pacotes, em vez de analisar cada pacote individualmente.

6. Proposta

Este trabalho tem por objetivo o estudo, a prototipação e a avaliação de diferentes detectores de ataques, baseados em fluxo, implementados em ambiente P4. Para apoiar esse objetivo, visa-se desenvolver um ambiente de experimentação onde será possível caracterizar e implementar esses detectores. Além disso, com o objetivo de avaliar a eficiência com que ataques são detectados, será possível a configuração de diversas topologias e *workloads*. A Figura 4 ilustra o ambiente de experimentação proposto.

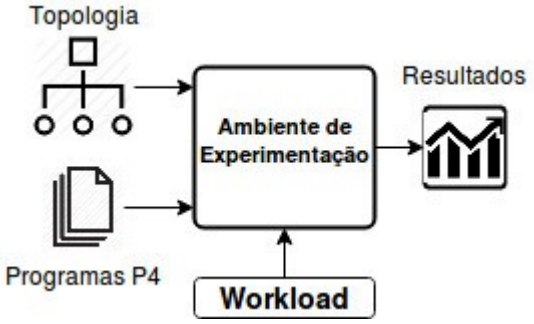


Figura 4. Estratégia proposta para avaliação de diferentes detectores.

Como mostrado na figura acima, o ambiente proposto possui três entradas e uma saída. A primeira entrada é a topologia da rede na qual o prolema será estudado. A segunda entrada refere-se a uma biblioteca de programas P4 que será implementada para a detecção de um conjunto de ataques baseados em padrões de fluxo, conforme Figura 1. Os diferentes tipos de ataques a serem detectados poderá ser configurado, de acordo com as necessidades do experimento. Finalmente, a terceira entrada trata-se de diferentes *workloads*, também configuráveis, que serão implementados para simular um ambiente de ataques real. Tais parâmetros configuráveis podem ser diversos tipos, como: número de pacotes por segundo, número de fluxos existentes, tamanho de pacotes, etc.

Por fim, o ambiente de experimentação vai gerar como saída resultados sobre a eficiência do experimento. Essa eficiência poderá ser medida através de diferentes métricas, como: número de falsos negativos (porcentagem de ataques que não são detectados), número de falsos positivos (porcentagem de fluxos erroneamente reportados como ataques), tempo decorrido entre o início de um ataque e sua respectiva detecção, utilização de memória no *switch*, entre outras. Além disso, para o Trabalho de Graduação 2 será feita um estudo sobre os diversos experimentos a serem realizados, contendo uma aprofundada análise sobre as métricas escolhidas e os diferentes *trade-*

offs entre elas.

7. Metodologia

A metodologia a ser aplicada para o desenvolvimento do Trabalho de Graduação 2 é apresentada a seguir. Até o presente momento, as etapas 1 e 2 já foram concluídas.

1. Estudo: etapa que consiste na melhor compreensão do problema através da análise bibliográfica do estado da arte relativo a técnicas de detecção de ameaças de segurança e como elas podem ser implementadas utilizando plano de dados programáveis.
2. Ambientação: etapa composta pela adaptação aos ambientes utilizados na etapa de Projeto. Realiza-se a instalação, aclimatação e compreensão das ferramentas de desenvolvimento, como *Mininet* e P4.
3. Projeto: etapas na qual são tomadas decisões relativas aos algoritmos que serão implementados, bem como o resultado esperado.
4. Implementação: etapa de desenvolvimento da aplicação propriamente dita.
5. Avaliação: etapa final destinada a testes e validação da solução proposta, além da comparação dos resultados obtidos com os resultados estimados na etapa de Projeto.

8. Cronograma

Com base na metodologia apresentada, o cronograma de desenvolvimento do Trabalho de Graduação compreende as seguintes tarefas:

1. Caracterização dos ataques a serem detectados.
2. Desenvolvimento do projeto dos algoritmos.
3. Integração das tecnologias desenvolvidas em um único ambiente parametrizável.
4. Avaliação experimental em ambiente simulado.
5. Redação do Trabalho de Graduação 2.
6. Apresentação do Trabalho de Graduação 2.

Tarefa	2017							2018
	Jun	Jul	Ago	Set	Out	Nov	Dez	Jan
1	**X**	**X**						
2		**X**	**X**	**X**				
3				**X**	**X**	**X**		
4					**X**	**X**	**X**	
5				**X**	**X**	**X**	**X**	
6								**X**

Tabela 1. Cronograma de atividades

9. Considerações Finais

Este trabalho apresentou uma breve análise sobre metodologia para a identificação de ameaças na rede. Além disso, foram apresentadas classificações e taxonomias sobre o tema e também técnicas para detecção de ataques de diferentes tipos como DDoS, *worms* e *scans*. Foram apresentados ainda conceitos básicos de planos de dados programáveis, enfatizando uma nova linguagem, chamada P4, criada para programar dispositivos de encaminhamento.

Finalmente, foi apresentada a proposta do Trabalho de Graduação 2. O plano prevê o desenvolvimento de um ambiente de experimentação configurável capaz de detectar um conjunto de ataques e gerar uma análise sobre a acurácia dessa detecção. Por fim, objetiva-se um estudo sobre os diversos experimentos a serem realizados e a comparação entre os diferentes resultados obtidos através da solução proposta.

Referências

Axelsson, S. (2000). Intrusion Detection Systems?: A Survey and Taxonomy. *Computer Engineering*, 1–27.

Bosshart, P., Varghese, G., Walker, D., Daly, D., Gibb, G., Izzard, M., … Vahdat, A. (2014). P4: Programming Protocol-Independent lPacket Processors. *ACM SIGCOMM Computer Communication Review*, *44*(3), 87–95.

Debar, H., Dacier, M., & Wespi, A. (1999). Towards a taxonomy of intrusion-detection systems. *Computer Networks*, *31*(8), 805–822.

Dübendorfer, T., & Plattner, B. (2005). Host behaviour based early detection of worm outbreaks in internet backbones. In *Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE* (Vol. 2005, p. 166–171).

Gao, Y., Li, Z., & Chen, Y. (2006). A DoS resilient flow-level intrusion detection approach for High-speed networks. In *Proceedings - International Conference on Distributed Computing Systems* (Vol. 2006).

Haddadi, F., Morgan, J., Filho, E. G., & Zincir-Heywood, A. N. (2014). Botnet behaviour analysis using IP flows: With http filters using classifiers. In *Proceedings - 2014 IEEE 28th International Conference on Advanced Information Networking and Applications Workshops, IEEE WAINA 2014* (p. 7–12).

Kim, M.-S. K. M.-S., Kong, H.-J. K. H.-J., Hong, S.-C. H. S.-C., Chung, S.-H. C. S.-H., & Hong, J. W. (2004). A flow-based method for abnormal network traffic detection. *2004 IEEE/IFIP Network Operations and Management Symposium (IEEE Cat. No.04CH37507)*, *1*, 1–14.

Li, Y., Miao, R., Kim, C., & Yu, M. (2016). FlowRadar: A Better NetFlow for Data Centers. *Nsdi '16*, 311–324.

Munz, G., & Carle, G. (2007). Real-time Analysis of Flow Data for Network Attack Detection. In *2007 10th IFIP/IEEE International Symposium on Integrated Network Management* (p. 100–108).

Popescu, D. A., Antichi, G., & Moore, A. W. (2017). Enabling Fast Hierarchical Heavy Hitter Detection Using Programmable Data Planes. In *Proceedings of the Symposium on SDN Research* (p. 191–192). New York, NY, USA: ACM.

Sivaraman, A., Cheung, A., Budiu, M., Kim, C., Alizadeh, M., Balakrishnan, H., … Licking, S. (2016). Packet Transactions: High-Level Programming for Line-Rate Switches. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference - SIGCOMM '16* (p. 15–28).

Sivaraman, V., Narayana, S., Rottenstreich, O., Muthukrishnan, S., & Rexford, J. (2017). HashPipe: Heavy Hitter Detection Entirely in the Data Plane. In *Proceedings of the Symposium on SDN Research - SOSR '17* (p. 164–176).

Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., & Stiller, B. (2010). An overview of IP flow-based intrusion detection. *IEEE Communications Surveys and Tutorials*, *12*(3), 343–356.

Wagner, A., & Plattner, B. (2005). Entropy based worm and anomaly detection in fast IP networks. In *Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE* (Vol. 2005, p. 172–177).

Zhao, Q., Xu, J., & Kumar, A. (2006). Detection of super sources and destinations in high-speed networks: Algorithms, analysis and evaluation. *IEEE Journal on Selected Areas in Communications*, *24*(10), 1840–1852.