Sistema de proteção contra ataques de HTTP *flood* usando Redes Definidas por Software

Diego S. M. Gonçalves[1,3], Rodrigo S. Couto[2], Marcelo G. Rubinstein[1] *

[1] Universidade do Estado do Rio de Janeiro - PEL/DETEL/FEN
[2]Universidade Federal do Rio de Janeiro - PEE/COPPE/GTA
[3]Grupo Globo

dstelman@g.globo,rodrigo@gta.ufrj.br,rubi@uerj.br

Abstract. *This paper proposes a protection system against HTTP flood attacks, based on Software Defined Networking (SDN). As these attacks are difficult to detect because they use valid requests, the system considers all suspicious requests in the first moment. When using HTTP protocol redirection associated with the flexibility of these networks, the system guarantees protection against this attack type. In addition, a collaboration between Internet autonomous systems is proposed to prevent network collapse during the attacks. The performance evaluation results show a reduction of 65.32% in CPU consumption of the SDN controller and a drop in latency perceived by customers from 6 s to 400 ms.*

Resumo. *Este trabalho propõe um sistema de proteção contra ataques de HTTP flood, baseado em redes definidas por software (SDN). Como esses ataques são de difícil detecção por utilizarem requisições válidas, o sistema considera todas as requisições suspeitas em um primeiro momento. Ao utilizar o redirecionamento do protocolo HTTP associado à flexibilidade dessas redes, o sistema garante proteção contra esse tipo de ataque. Além disso, é proposta uma colaboração, entre sistemas autônomos da Internet, para evitar que a rede entre em colapso durante os ataques. Os resultados da avaliação de desempenho mostram uma queda no consumo de CPU do controlador SDN de 65,32% e uma queda de latência percebida pelos clientes de 6 s para 400 ms.*

1. Introdução

Ataques distribuídos por negação de serviço (DDoS - *Distributed Denial of Service*) consistem em impedir que usuários tenham acesso a um determinado serviço. Ataques DDoS visam prejudicar tanto serviços não comerciais quanto aplicações de negócios críticas, nas quais as organizações confiam para gerenciar operações diárias, como servidores de e-mail e automação de vendas [NETSCOUT, 2020]. O estudo presente em [Institute, 2015] mostra que as 641 empresas pesquisadas relataram em média perdas de 1,5 milhão de dólares em um ano em função desse tipo de ataque.

Um tipo de ataque DDoS bastante conhecido é o ataque de inundação HTTP (*HyperText Transfer Protocol*) ou HTTP *flood*. Esse ataque geralmente depende de uma *botnet*, que é um grupo de equipamentos conectados à Internet que foram comprometidos

*O trabalho foi realizado com recursos do CNPq, da FAPERJ e da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), auxílio no. 2015/24494-8.

pelo uso de *malware* e são usados para realizar ataques a um determinado alvo na Internet. Além disso, esse ataque é de difícil detecção pois é realizado com requisições HTTP válidas, fazendo com que servidores web atendam a todas as requisições como sendo normais. Uma maneira de lidar com o ataque HTTP *flood* é considerar as requisições como suspeitas e redirecioná-las para a aplicação web desejada. Esse redirecionamento é feito através de um IP exposto na Internet que não é o IP da aplicação desejada. Esse IP exposto hospeda um serviço que redireciona os clientes para o IP com a aplicação web deseja. Dessa forma, os cliente válidos conseguem o acesso pois seguem as instruções de redirecionamento, ao contrário das requisições maliciosas que não seguem essas mesmas instruções [Ltd., 2020].

Este artigo propõe uma solução que protege, usando SDN, um servidor web de ataques DDoS HTTP *flood*. A solução proposta combina as características flexivas das redes SDN com o redirecionamento HTTP. Durante um ataque, mudanças rápidas em uma rede são necessárias e a flexibilidade fornecida pelas redes SDN, como a sua programabilidade, permitem que essas mudanças sejam realizadas. Dessa maneira, os ataques DDoS do tipo HTTP *flood* são tratados e as futuras requisições de clientes maliciosos, a contar a partir da primeira requisição, são bloqueadas na rede. Além disso, para que o controle da rede SDN não seja perdido em função do comprometimento dos recursos da CPU do controlador, uma VPN (*Virtual Private Network*) conecta o sistema a outros sistemas colaboradores que se encontram em outros ASs (*Autonomous Systems*). Isso permite o compartilhamento de recursos de processamento entre todos os controladores. O modelo de negócios do sistema está fora do escopo deste trabalho, mas a estrutura de colaboração pode ser disponibilizada por empresas que, por meio de contratos preestabelecidos, são remuneradas pelo volume de tráfego redirecionado ou pelo tempo de utilização da estrutura, por exemplo. Neste trabalho, a solução é implementada em um ambiente virtual de testes, objetivando a avaliação do seu desempenho. Os resultados mostram uma queda no consumo de CPU do servidor do controlador SDN de 65,32% e uma queda da latência percebida pelos clientes de 6 s para 400 ms.

O restante do artigo está organizado da seguinte forma. A Seção 2 discute os trabalhos relacionados. O sistema proposto é descrito na Seção 3, seu funcionamento é apresentado na Seção 4 e avaliado na Seção 5. Finalmente, a Seção 6 apresenta as principais conclusões e sugestões para trabalhos futuros.

2. Trabalhos Relacionados

Diversos autores utilizam as propriedades das redes SDN para propor soluções de gerenciamento de rede e de segurança. Em [Dao et al., 2015], propõe-se um sistema no qual todas as requisições que chegam até o controlador têm seus IPs registrados. Assim, um contador associado a cada um desses IPs é utilizado para registrar a quantidade de pacotes enviados pelos mesmos. O controlador insere fluxos para as novas entradas com os seus *hard timeouts* e *idle timeouts* com intervalos de tempo menores do que os valores encontrados em fluxos já validados como legítimos. Essa abordagem é realizada para fluxos novos durarem um tempo menor, já que não são confiáveis inicialmente. Se a quantidade de pacotes e conexões dentro do tempo estabelecido não estiverem de acordo com o que o sistema considera normal, o IP é classificado como atacante e um fluxo de bloqueio é inserido. Diferentemente de [Dao et al., 2015], o sistema proposto neste trabalho classifica todos os IPs como suspeitos em um primeiro momento para depois validar, ou

não, esses IPs em função dos seus comportamentos sem a necessidade de monitoração do cliente por parte do sistema. Além disso, os autores não se preocupam com a estabilidade do controlador durante um ataque.

Baseando-se na flexibilidade fornecida pelas redes SDN, [Lim et al., 2014] apresentam uma solução para ataques em servidores web. Quando novas requisições chegam até os servidores, o controlador insere fluxos permissivos para que cliente e servidor se comuniquem. Quando o sistema encontra-se sob ataque, o sistema muda o IP do servidor, insere fluxos permissivos para o novo IP, finaliza as conexões antigas e envia aos clientes um redirecionamento para o novo IP. Além disso, cada cliente precisa responder um CAPTCHA. Caso o cliente não seja capaz de seguir o redirecionamento e não responder o CAPTCHA, um fluxo de bloqueio é inserido pelo sistema para o IP solicitante. Todavia, o trabalho não propõe soluções de segurança para o controlador, que terá seu desempenho afetado durante ataques. Para preencher essa lacuna, este trabalho propõe a colaboração entre sistemas através de VPNs que conectam o sistema criado a outros sistemas colaboradores localizados em outros ASs. Isso evita o comprometimento do controlador que causaria um colapso no sistema com sua perda.

A colaboração entre ASs é também proposta por [Hameed e Khan, 2017]. Nesse trabalho, os autores propõem um esquema colaborativo de mitigação de ataques DDoS usando SDN, no qual diferentes controladores também encontram-se em ASs distintos. Esses controladores se comunicam através de um protocolo que permite que informações sobre ataques detectados e ações de bloqueio sejam compartilhadas entre todos os controladores. Quando o sistema detecta IPs suspeitos, os controladores atualizam os fluxos de bloqueio em todo o sistema. O sistema proposto neste trabalho utiliza VPNs para implementar a colaboração entre ASs. Assim, diferentemente de [Hameed e Khan, 2017], não há necessidade de emprego de um protocolo próprio. A proposta deste trabalho usa VPNs que são mais fáceis de serem implantadas, pois não precisam de soluções específicas.

Em [Wang et al., 2015], a preocupação com o comprometimento do controlador durante um ataque DDoS é abordada por meio da inserção de fluxos proativos nos comutadores. Esses fluxos são inseridos utilizando um algoritmo desenvolvido pelos autores e são estimados a partir da capacidade do controlador de lidar com os fluxos em um instante futuro. Essa ação evita um alto consumo de recursos do controlador, afinal, quando não existe um fluxo escrito para um determinado pacote, o comutador envia mensagens *packet-in* para o controlador a fim de configurar um novo fluxo. Durante um ataque, muitas mensagens *packet-in* seriam processadas gerando problemas mas, com os fluxos proativos já escritos, esse problema não ocorre. Assim como em [Wang et al., 2015], o modo proativo é usado neste trabalho, porém há algumas diferenças entre as propostas. Na proposta deste trabalho já existe um fluxo permissivo no comutador evitando o envio de mensagens *packet-in* ao controlador. Além disso, a preocupação com a integridade do controlador se dá através da distribuição de requisições com outros controladores.

O trabalho de [Dridi e Zhani, 2016] propõe um sistema de proteção que evita a sobrecarga do controlador, o congestionamento do enlace entre o controlador e os comutadores, e a saturação das tabelas de fluxo. O sistema coleta estatísticas por meio de um IDS (*Intrusion Detection System*) e determina o *hard timeout* de cada fluxo baseado na probabilidade de ameaça representada por esses fluxos. Além disso, reduz o número de fluxos suspeitos que permanecem na tabela de fluxos, ao agregá-los quando possuem pro-

priedades semelhantes (p.ex., mesmos origem e destino). Esses fluxos são mantidos para que, em casos de falsos positivos, o cliente consiga acesso. Contudo, esse cliente tem um atraso maior, pois esses fluxos agregados são destinados a enlaces de pior desempenho. Em uma abordagem mais simples que [Dridi e Zhani, 2016], o sistema proposto neste trabalho trata todas as requisições como suspeitas sem considerar falsos positivos. Porém, em poucos segundos essas requisições são classificadas como maliciosas ou legítimas. Sendo assim, fluxos maliciosos não permanecem na tabela do comutador por longos períodos.

3. Arquitetura do Sistema Proposto

O sistema é composto de um Sistema Local (SL) que pode trabalhar com um ou mais ASs Colaboradores (ASCs), como exemplificado na Figura 1 para o caso de um ASC. O Sistema Local é o sistema que abriga o servidor com a Aplicação web e contém um Armazenador, um Redirecionador e uma rede SDN composta por um Controlador e um comutador virtual (OVS na implementação realizada). É de se destacar que o Sistema Local, sem um ASC funcionando em conjunto, é capaz de se defender normalmente. A colaboração de outro AS é uma extensão do sistema que visa proteger os recursos do Controlador do SL em situações de ataque e reduzir a latência percebida pelos clientes nessas situações. Um ASC é uma estrutura que contém uma rede SDN composta por um Controlador, um OVS e um Tradutor. Os controladores dos ASCs e do SL, o Redirecionador e o Armazenador se comunicam através de uma VPN denominada Controladores. Da mesma forma, o Tradutor e a Aplicação web se comunicam através da VPN Tradutor. A seguir são descritos os principais componentes do sistema proposto, apresentado na Figura 1 e suas funcionalidades. Mais adiante, na Seção 4, o funcionamento do sistema é detalhado.

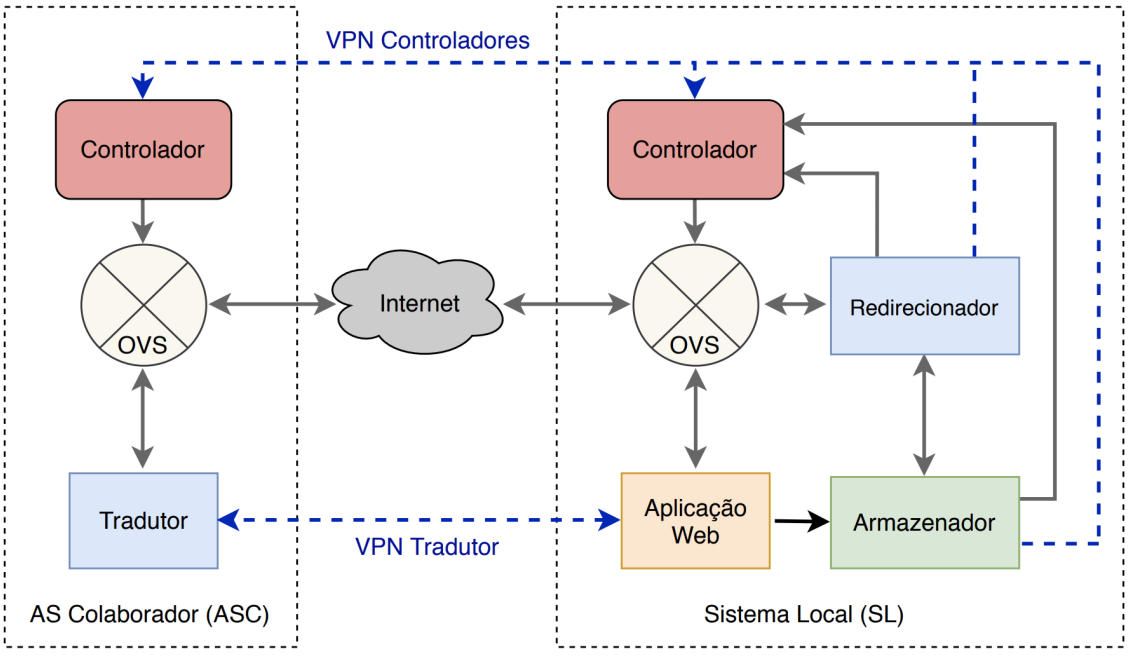

Figura 1. Arquitetura do sistema proposto.

3.1. Sistema Local

O Sistema Local possui essa denominação pois abriga o serviço a ser protegido dos ataques vindos de *botnets*. O SL é a parte central de todo o sistema e seus blocos são

descritos a seguir.

O Redirecionador é um servidor web que redireciona todas as requisições para determinados destinos específicos, além de ser capaz de se comunicar com o Controlador e o Armazenador. Esse bloco possui seu IP exposto na Internet por DNS. Dessa forma, é o ponto inicial de contato com o usuário da Aplicação web. Para tal, um fluxo permissivo para o Redirecionador é previamente instalado no OVS do SL. Para cada requisição recebida pelo Redirecionador, registram-se no Armazenador informações sobre a requisição. Após essa escrita, um fluxo permissivo para o IP de origem da requisição é escrito pelo Controlador no OVS. Esse fluxo permite que o IP de origem da requisição seja capaz de alcançar o IP da aplicação web em uma requisição futura e, assim, o cliente consegue acesso ao serviço desejado. Após a escrita do fluxo no OVS, o cliente é redirecionado para o IP da Aplicação web por meio da resposta 302 *status code* do protocolo HTTP. Quando o cliente recebe a resposta da sua requisição ao sistema com o código 302, o cabeçalho HTTP contém um campo denominado *Location*. Esse campo possui o IP a ser acessado em uma requisição futura pelo cliente. No caso do sistema proposto, o IP do campo *Location* é o IP da Aplicação web.

A ideia de utilizar o redirecionamento é fornecer um ”desafio” HTTP para os usuários que pretendem acessar a aplicação web. É esperado que os usuários legítimos superem o desafio, redirecionando suas requisições após receber a mensagem 302. Assume-se, neste trabalho, que os usuários maliciosos não implementam o protocolo HTTP completo e, assim, não são capazes de superar o mesmo desafio, afinal as *botnets* não são programadas para seguir o redirecionamento enviado pelo servidor sob ataque [Lim et al., 2014]. Algumas ferramentas e *botnets* mais sofisticadas podem investir na superação desse desafio até mesmo utilizando navegadores web, porém nesse caso a capacidade de ataque será diminuída [Ltd., 2020].

Quando um cliente consegue realizar requisições para a Aplicação web, sabe-se que o mesmo já realizou requisições para o Redirecionador, já foi cadastrado no Armazenador e possui um fluxo permissivo instalado no OVS para o acesso ao seu destino final (isto é, a Aplicação web). Na primeira requisição realizada com sucesso, a Aplicação web atualiza o estado do IP de origem do cliente no Armazenador. Essa permissão é concedida pois o sistema classificou o cliente como legítimo; afinal, todas as etapas de validação anteriores foram cumpridas pelo cliente. Caso o IP de origem de uma requisição seja de uma *botnet*, essa atualização no Armazenador não ocorre, levando o sistema a classificar o cliente como atacante.

O Armazenador registra informações referentes às requisições de entrada no sistema, além de ser capaz de se comunicar com o Controlador. Sempre que uma requisição é recebida, o Redirecionador realiza uma consulta ao Armazenador para verificar se o IP referente à requisição de entrada é conhecido pelo sistema. Dependendo do estado que o IP é classificado no Armazenador, o Redirecionador tomará diferentes decisões. Caso a requisição seja realizada por um IP classificado como malicioso, essa é ignorada. Caso contrário, o Redirecionador envia uma resposta com o redirecionamento a ser seguido pelo cliente. Uma outra função do Armazenador é monitorar os seus registros para bloquear IPs suspeitos de serem maliciosos. Caso o Armazenador contenha em seus registros IPs que não foram validados pela Aplicação web após um intervalo de tempo predeterminado, esses IPs são classificados como maliciosos.

A aplicação do Armazenador utiliza uma API (*Application Programming Interface*) com o *framework* Flask [Flask, 2019]. Suas informações são registradas utilizando um armazenador de dados em memória conhecido como Redis [Redis, 2019]. O Redis pode atuar como um banco de dados ou também como um *cache*. Essa solução é utilizada neste trabalho devido à sua baixa latência, visto que realiza armazenamento em memória, e à sua facilidade de uso [Amazon Web Services, 2020].

Cada requisição é registrada no Armazenador com os seguintes campos:

- IP - contém o IP de origem da requisição. Esse IP é o identificador único para um determinado cliente, coletado no momento da sua primeira requisição;
- CONTROLE - contém as informações do estado, ID do Controlador e tempo para cada IP registrado.

A Tabela 1 mostra um exemplo de informações registradas no Armazenador. Cada IP é associado a um conjunto de caracteres que guarda as informações de controle separadas pelo caractere "_". O primeiro campo de caracteres é referente ao estado de um determinado IP. O estado contém os valores definidos pelo sistema para classificar um cliente. Os valores 0, 1 e 2 classificam, respectivamente, um cliente como não validado, validado e atacante. Quando o IP é definido como atacante, somente o valor do estado fica registrado pois essa informação é salva apenas para registro; afinal, o IP já encontra-se bloqueado na entrada do sistema. O segundo campo de caracteres contém o ID do AS Colaborador para o qual a requisição deverá ser redirecionado, se for o caso de estar usando um AS Colaborativo. Caso o sistema não esteja utilizando colaboração, o ID será 0. O sistema é capaz de saber, através desse ID, qual controlador utilizar e para qual IP as requisições devem ser redirecionadas, como descrito na Seção 3.2. O terceiro campo de caracteres contém o valor do momento no qual o IP foi registrado no Armazenador. Esse valor é salvo em *Unix Time* e, utilizando esse valor, o Armazenador é capaz de calcular há quanto tempo a requisição encontra-se cadastrada.

Tabela 1. Exemplo de informações de armazenamento.

IP	CONTROLE
195.3.1.4	2
12.44.32.199	1_1_156969833
178.43.55.12	0_1_156969839

Quando o SL opera com ASs Colaboradores, o sistema passa a distribuir as requisições de entrada entre ASs na Internet. Essa colaboração é proposta devido aos limites de processamento do Controlador do SL, já que mitigar ataques DDoS normalmente exige muita capacidade de processamento. Assim, dividir requisições entre diversos ASs aumenta a quantidade global de recursos de processamento disponíveis para o AS atacado.

3.2. AS Colaborador

Quando o sistema utiliza ASCs, os mesmos recebem requisições redirecionadas pelo SL através do algoritmo de balanceamento *round-robin*. O sistema proposto pode operar com quantos ASCs forem possíveis. Esses ASCs são referenciados no segundo campo de caracteres encontrados na Tabela 1. Além do Controlador e do OVS, cada ASC possui um bloco denominado Tradutor. Esse bloco é necessário para permitir que a

Aplicação web, que encontra-se em um determinado AS, seja acessada diretamente através de diferentes ASs na Internet. O Tradutor recebe as requisições redirecionadas pelo SL e as encaminha para a Aplicação web por meio da VPN Tradutor. O sistema assume que todo o tráfego que percorre essa VPN é confiável, ou seja, acessa a Aplicação web diretamente, sem intermédio do Redirecionador. Além disso, somente requisições que acessaram previamente o Redirecionador terão fluxos permissivos escritos para acesso à Aplicação web via VPN.

Quando uma requisição recebida no SL é redirecionada para um ASC, o Redirecionador registra seus dados no Armazenador. Utilizando a VPN Controladores, o Redirecionador se comunica com o Controlador do ASC para inserir um fluxo permissivo no OVS do ASC. Nesse caso, o IP de destino desse fluxo não é o IP da Aplicação web, mas o IP do Tradutor.

A Figura 3.2 apresenta um exemplo de funcionamento do Tradutor. Quando o fluxo permissivo é inserido no OVS do ASC para o IP de destino 107.13.1.2 (endereço do Tradutor), a requisição redirecionada pelo SL acessa esse IP e o Tradutor realiza um NAT (*Network Address Translation*) no IP de origem para o IP 10.10.1.2, que é o IP do Tradutor na VPN do exemplo da Figura 3.2. O IP de destino, que consiste no IP da Aplicação web na VPN, também é traduzido para um IP na faixa 10.10.1.0/24.

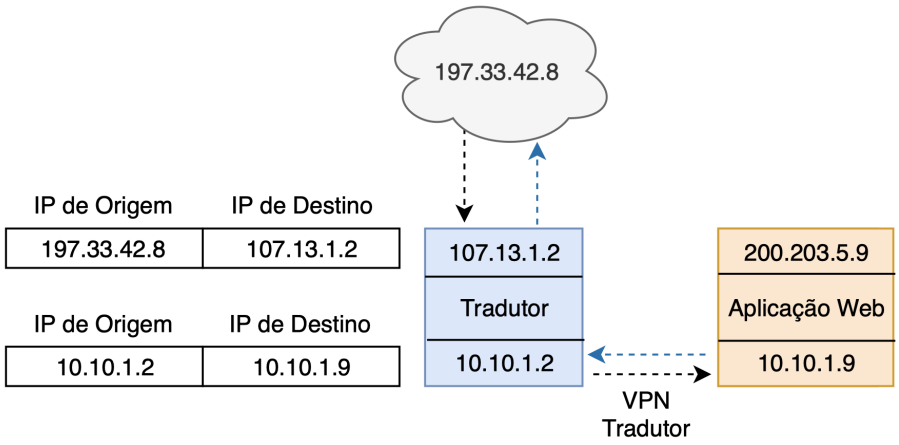

Figura 2. Exemplo de funcionamento da estrutura do AS Colaborador.

Da mesma forma que para o SL, caso nenhuma requisição atinja a Aplicação web, o Armazenador remove o fluxo permissivo no OVS do ASC, atualiza o estado do IP de origem para atacante e insere um fluxo de bloqueio no OVS do SL.

4. Funcionamento do Sistema

O sistema proposto pode funcionar sem colaboração de outros ASs ou com essa colaboração. Cada um desses modos é descrito a seguir.

4.1. Modo sem colaboração

Neste modo, apenas o SL da Figura 1 é ativado. Inicialmente, o OVS possui apenas um fluxo permissivo para que todo e qualquer endereço IP consiga acesso ao Redirecionador. Quando o Redirecionador recebe uma requisição qualquer da Internet, imediatamente

essa requisição é cadastrada no Armazenador. O cadastro dessa nova entrada tem o campo estado com o valor não validado, ou seja, é uma entrada que ainda precisa ser verificada pelo sistema. Após esse cadastro, um fluxo permissivo no OVS é inserido para que o IP de origem dessa requisição seja capaz de acessar a Aplicação web protegida por meio do redirecionamento. Esse fluxo pode ser mantido como permissivo ou bloqueado, dependendo da resposta do cliente ao redirecionamento.

Um ponto importante a se observar é a atualização do campo estado no Armazenador. Caso uma requisição já esteja cadastrada no Armazenador com o valor do campo estado igual a 1 (validada), o Redirecionador apenas envia o redirecionamento para o cliente com destino ao IP cadastrado no campo aplicação. Assim, novas escritas de fluxos no OVS não são realizadas pois o fluxo permissivo para esse cliente já foi escrito por meio de uma requisição prévia. O campo estado com valor 2 (ataque) é utilizado meramente para registro. Afinal, assim que o Redirecionador escreve um fluxo de bloqueio para um determinado IP e atualiza o campo estado para 2, esse IP não consegue mais realizar requisições para o Redirecionador. Dessa forma, o seu cadastro no Armazenador não é mais atualizado.

O Armazenador também busca requisições que tenham seu campo estado ainda não validado. Periodicamente, todo campo estado com valor 0 tem seu tempo de escrita no Armazenador verificado através dos valores do terceiro campo de caracteres. Esse campo, como já mencionado, contém o intervalo de tempo no qual o IP foi registrado (em *Unix Time*). Se em 10 s a Aplicação web não atualiza o campo estado para 1, o Armazenador assume que a requisição não foi capaz de seguir o redirecionamento enviado previamente, tratando-se de uma requisição maligna. Após isso, o campo estado é atualizado com o valor 2 (atacante) e um fluxo de bloqueio é inserido no OVS. O fluxo permissivo para esse IP de origem é então removido do OVS. Por outro lado, se o cliente acessa o conteúdo da Aplicação web, o Armazenador tem o estado dessa requisição atualizado para validado e o fluxo permissivo é mantido.

4.2. Modo com colaboração

Como no modo sem colaboração, o OVS do SL inicialmente possui apenas fluxos permissivos para que qualquer endereço IP consiga acesso ao Redirecionador. Além do IP da aplicação web e do Controlador do SL, o Redirecionador possui um conhecimento prévio dos IPs de todos os Tradutores e Controladores dos ASCs. Dessa maneira, quando o Redirecionador recebe requisições, um cadastro é realizado no Armazenador com as informações referentes a cada requisição. Após o cadastro, cada requisição terá um fluxo permissivo escrito em algum OVS do sistema. A distribuição de endereços IPs pelos OVSs é realizada pelo Redirecionador de forma balanceada, utilizando a política *round-robin*.

Após o IP ser cadastrado e o fluxo inserido, um HTTP *redirect* 302 é enviado para o cliente, redirecionando para o IP do Tradutor do ASC selecionado. Da mesma forma que no modo sem colaboração, o Armazenador, periodicamente, verifica quais entradas estão em estado ainda não validado por mais de 10 s. Caso alguma requisição permaneça no estado não validado por mais de 10 s, um fluxo de bloqueio é inserido no OVS e o fluxo permissivo para acesso à aplicação é removido. Essa estratégia é realizada para qualquer AS no qual o fluxo se encontra. Caso o cliente acesse a Aplicação web, seu estado se torna validado e o fluxo permissivo se mantém. Futuras requisições para esse fluxo são redirecionadas para o ASC correspondente.

O cliente, seguindo as instruções do Redirecionador, realiza requisições para o IP do Tradutor. O Tradutor, por sua vez, utiliza NAT para encaminhar essa solicitação para a VPN de acesso à Aplicação web e assim o cliente consegue acesso à aplicação.

5. Avaliação de Desempenho do Sistema Proposto

O desempenho do sistema proposto é avaliado por meio de três cenários. No primeiro, a latência das requisições para a aplicação web é medida sem a atuação do sistema proposto. No segundo cenário, o funcionamento do sistema é avaliado sem a colaboração de outro AS. Nesse caso, medem-se o consumo de CPU do controlador do SL e a latência das requisições para a aplicação tanto em casos de ataque quanto em casos sem ataque. O terceiro cenário realiza as mesmas medidas do segundo, mas utilizando ASCs. Apresentam-se a seguir as configurações empregadas e os resultados obtidos.

5.1. Configurações

Todos os experimentos utilizam máquinas virtuais (*Virtual Machines* - VMs) do VirtualBox 6.0 [VirtualBox, 2019], instalado em um MacBook Pro com 16 GB de RAM e um processador Intel(R) Core(TM) i5-3210 CPU @ 2,50 GHz. O sistema proposto é avaliado no emulador de SDN Mininet [Mininet, 2019]. Para os ataques, uma *botnet* é simulada pela ferramenta Bonesi [Bonesi, 2019]. Neste trabalho, esse simulador realiza ataques HTTP *flood* a uma taxa de 100 requisições por segundo.

O ambiente configurado possui um SL e até seis ASCs. Diferentes VMs foram criadas para cada um dos Controladores, o Redirecionador, a Aplicação web a ser protegida e o Armazenador. Assim, são utilizadas até 12 VMs para a realização dos experimentos, listadas a seguir: sete VMs para os Controladores; uma VM para o Redirecionador; uma VM para a Aplicação web; uma VM para o Armazenador; uma VM para o ambiente Mininet; uma VM para o simulador de *botnet* Bonesi.

Todas as VMs utilizam o sistema operacional Ubuntu 16.04 LTS e uma interface virtual de rede de 1 Gbit/s. Todos os links do ambiente Mininet utilizados foram de 1 Gbit/s com 1 ms de atraso. As VMs dos Controladores possuem, cada uma, 512 MB de RAM. As VMs referentes ao Armazenador, à Aplicação web e ao simulador Bonesi possuem, cada uma, 1 GB de RAM. A VM do Redirecionador e a VM do Mininet possuem 2 GB de RAM.

Este trabalho utiliza o Ryu [Ryu, 2019] como controlador SDN, por seu desempenho satisfatório [Khondoker et al., 2014] e por ser uma solução gratuita e de código aberto. Além disso, o Ryu possui uma API escrita em Python. Isso facilitou o desenvolvimento, já que tanto a API do Mininet quanto o código do sistema proposto são escritos nessa linguagem[1].

Para avaliar o comportamento do sistema proposto, 50 clientes enviam requisições HTTP benignas para o SL. Um cliente é sorteado aleatoriamente usando uma distribuição uniforme entre 1 e 50 e envia uma quantidade de requisições por segundo também aleatória, seguindo uma distribuição uniforme com valores entre 5 e 50. Em seguida outro cliente é sorteado e envia as suas requisições ate completar o total de 50 clientes. Quando o sistema é atacado, paralelamente às requisições benignas, uma *botnet* simulada pela ferramenta Bonesi [Bonesi, 2019] realiza requisições a uma taxa de 100 requisições/s. Essa

[1]Os códigos do sistema proposto estão disponíveis em https://github.com/dstelman/sistema.git

metodologia é utilizada em todos os experimentos, objetivando uma comparação justa entre os cenários.

5.2. Resultados

A seguir, o desempenho da Aplicação web sem a proteção do sistema proposto é analisado. Seus resultados são usados como base para comparação com os resultados obtidos pelo sistema proposto. Posteriormente, os dois cenários com o sistema em funcionamento são avaliados. Os valores apresentados em todas as figuras de resultados são médias com intervalos de confiança de 95%.

5.2.1. Comportamento da Aplicação web sem a atuação do sistema

O cenário com a Aplicação web sem o sistema proposto atuando mostra o quanto o desempenho da Aplicação web, percebido pelos clientes, pode piorar durante um ataque. Para este cenário, a Figura 3 mostra a latência percebida pelos clientes da aplicação web quando a mesma lida com tráfego benigno e com tráfego de ataque. Cada amostra do experimento é a média da latência dos clientes. Analisando o gráfico, nota-se que a latência aumenta de alguns milissegundos para 6 s quando há ataque; ou seja, há um aumento significativo na latência percebida pelos clientes. De acordo com [Nngroup, 2019], a latência para servidores web não deve passar de 1 s e esse resultado mostra que o serviço oferecido aos clientes seria severamente prejudicado.

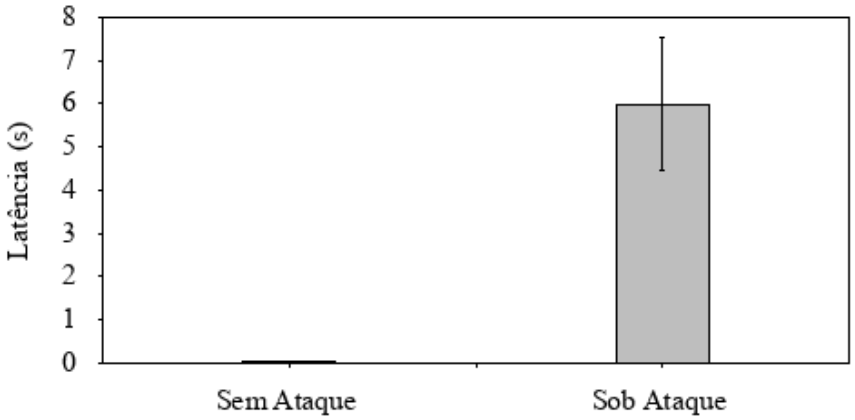

Figura 3. Latência percebida pelos clientes com a aplicação web desprotegida.

5.2.2. Comportamento da Aplicação web com atuação do sistema

O objetivo deste cenário é verificar o quanto o sistema é capaz de melhorar a latência das requisições dos clientes. A Figura 4 apresenta a latência percebida pelos clientes quando o sistema atua sem colaboração. Apesar de a Figura 4 possuir um eixo Y em escala diferente da usada na Figura 4, é possível comparar as duas numericamente. Nos resultados da Figura 3, a latência em situações sem ataque é de 40 ms, enquanto esse mesmo caso na Figura 4 apresenta uma latência de 111 ms; ou seja, o sistema proposto adiciona latência à aplicação em situações sem ataque. Isso se deve ao fato de ser

necessária uma maior interação entre os componentes do sistema, pois o serviço agora depende de requisições internas, tanto ao Armazenador quanto ao Controlador do SL. Em termos práticos, o aumento da latência é imperceptível para o usuário final, de acordo com o estudo de [Nngroup, 2019].

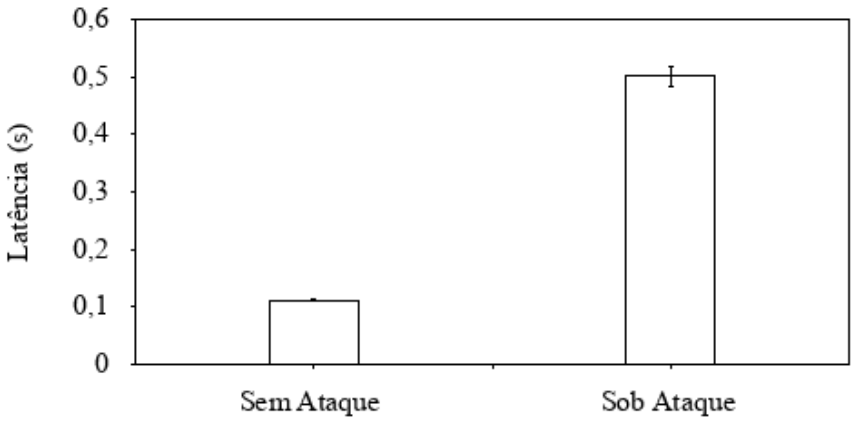

Figura 4. Latência percebida pelos clientes com o sistema sem colaboração.

A Figura 4 também apresenta uma queda de latência das requisições válidas durante um ataque, em comparação com os valores mostrados na Figura 3. A queda de 6 para 0,5 s mostra que o sistema é capaz de diminuir significativamente o tempo de resposta das requisições de usuários válidos durante uma situação de ataque. Esse cenário mostrou que o sistema é capaz de reduzir a latência percebida pelos usuários, podendo ser utilizado como uma opção contra ataques do tipo HTTP *flood* a uma Aplicação web na Internet.

Além da latência, o consumo de CPU do controlador do SL deve ser verificado pois, caso a CPU do controlador do SL sature, o sistema entra em colapso, não atendendo novas requisições dos usuários legítimos. Durante este experimento, a porcentagem de utilização da CPU da VM do Controlador do SL apresentou picos de 85,68%. Um ataque maior do que o utilizado nos experimentos pode fazer com que a utilização da CPU chegue a 100%, levando o sistema ao colapso. Assim, os próximos experimentos mostram como ASs Colaboradores podem ser utilizados para reduzir o uso de CPU do Controlador do SL.

5.2.3. Comportamento da Aplicação web com atuação do sistema e colaboração

O objetivo deste cenário é verificar o quanto o sistema é capaz de melhorar a latência das requisições quando ASs Colaboradores são utilizados. Além disso, analisa-se o quanto o uso de CPU do Controlador do SL pode ser reduzido com essa estratégia.

A Figura 5 apresenta a latência percebida pelos clientes legítimos em situações sem ataque quando varia-se o número de ASs Colaboradores. Para efeito de comparação, a figura também contém os valores de latência quando não há colaboração (zero ASCs), apresentados anteriormente nas Figuras 3 e 4. Como observado anteriormente, a latência aumenta quando o sistema passa a proteger a Aplicação web. Além disso, a latência au-

menta quando comparada com o sistema sem colaboração. Esse aumento se deve ao NAT realizado nos pacotes em cada ASC.

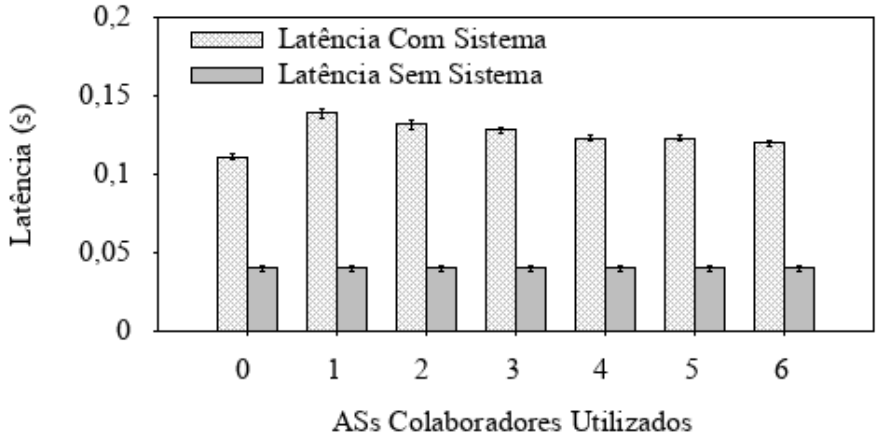

Figura 5. Latência percebida pelos clientes para diferentes números de ASs colaboradores, no caso sem ataque.

A Figura 6 apresenta os resultados referentes à latência percebida pelos clientes legítimos em situações de ataque. Na figura encontra-se também o valor da latência para o sistema operando sem colaboração para fins comparativos (ASC com valor 0). Os resultados mostram que essa latência diminui conforme o número de ASCs aumenta, ficando praticamente estável a partir de 4 ASCs.

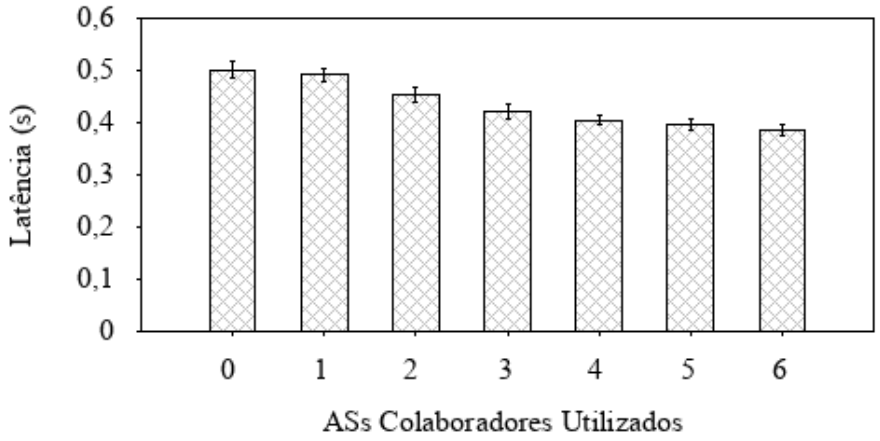

Figura 6. Latência percebida pelos clientes para diferentes números de ASs colaboradores no caso de ataque.

A Figura 7 apresenta o consumo de CPU da VM do Controlador do SL em situações de ataque. De acordo com esses resultados, o uso de CPU do Controlador do SL é reduzido em função do aumento de ASCs colaboradores inseridos no sistema. Além disso, a Figura 7 também mostra o consumo de CPU do Controlador do SL quando o mesmo não encontra-se em uso. Com o aumento das contribuições, o consumo de CPU do Controlador do SL aproxima-se do valor do consumo quando não se usa o Controlador do SL que é de 12%.

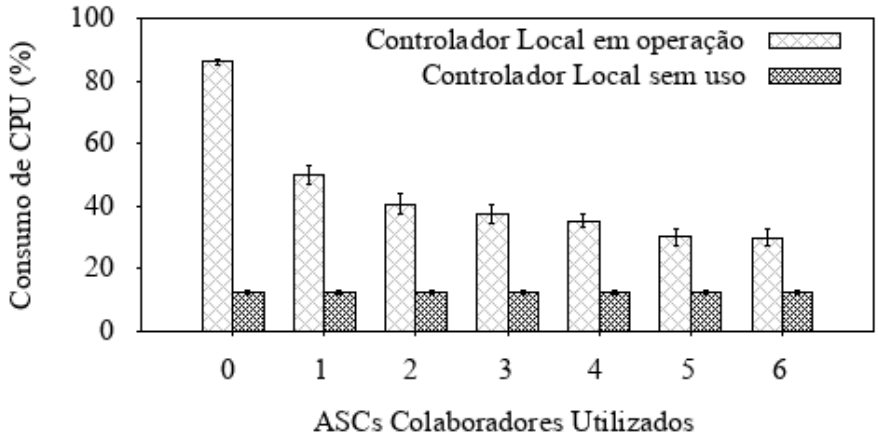

Figura 7. Consumo de CPU da VM do Controlador do SL com ataque.

Este cenário avaliado mostra que o sistema com cooperação é capaz de atuar contra ataques do tipo HTTP *flood* e de melhorar seu desempenho se comparado ao cenário sem colaboração. Outra vantagem de se utilizar a colaboração é obter uma redução do consumo da CPU do Controlador do SL. Isso se torna um artifício importante ao evitar o colapso do sistema como um todo, quando recebe uma elevada taxa de requisições.

6. Conclusão e Trabalhos Futuros

Este artigo propôs um sistema de proteção baseado em SDN para uma Aplicação web contra ataques de *botnets* do tipo HTTP *flood*. A solução é baseada em ASs Colaboradores conectados por VPNs e busca reduzir o uso de CPU do Controlador do Sistema Local. Para a colaboração, o sistema assume que todo o tráfego entre ASs, enviado pelos túneis de VPN, é confiável, não necessitando de tratamento contra ataques para os mesmos.

Os resultados das emulações mostram que, visando evitar a perda do Controlador do SL, a distribuição do tráfego ajuda a baixar o consumo de CPU do mesmo, evitando assim o colapso do sistema. Além disso, mostra-se que o sistema é capaz de diminuir a latência para requisições legítimas à medida que o número de ASs Colaboradores aumenta.

Como trabalhos futuros deseja-se virtualizar o Redirecionador para paralelizar o serviço, evitando assim que apenas uma única máquina processe todas as requisições. Isso também adiciona elasticidade ao serviço, introduzindo uma capacidade de mudança de suas características quando necessário. Também deseja-se medir na VPN a latência real entre ASs da Internet e o SL a fim de verificar a variação da latência em função das diferentes distâncias dos ASs para o SL. Além disso, o conceito de *threat intelligence* também pode ser aplicado. Esse conceito permite bloquear os IPs suspeitos de uma forma mais inteligente, na qual os IPs seriam classificados não apenas como maliciosos ou não maliciosos, mas através do uso de diferentes níveis de periculosidade.

Referências

Amazon Web Services, I. (2020). What is redis? https://aws.amazon.com/pt/elasticache/what-is-redis/.

Bonesi (2019). Bonesi. https://github.com/markus-go/bonesi.

Dao, N.-N., Park, J., Park, M. e Cho, S. (2015). A feasible method to combat against DDoS attack in SDN network. Em *IEEE International Conference on Information Networking (ICOIN)*, p. 309–311.

Dridi, L. e Zhani, M. F. (2016). SDN-guard: Dos attacks mitigation in SDN networks. Em *IEEE Cloud Networking (Cloudnet)*, p. 212–217.

Flask (2019). Flask. http://flask.palletsprojects.com/en/1.1.x/.

Hameed, S. e Khan, H. A. (2017). Leveraging SDN for collaborative DDoS mitigation. Em *IEEE Networked Systems (NetSys)*, p. 1–6.

Institute, P. (2015). The Cost of Denial-of-Services Attacks. Relatório técnico, Ponemon Institute LLC.

Khondoker, R., Zaalouk, A., Marx, R. e Bayarou, K. (2014). Feature-based comparison and selection of Software Defined Networking (SDN) controllers. Em *IEEE World Congress on Computer Applications and Information Systems (WCCAIS)*, p. 1–7.

Lim, S., Ha, J., Kim, H., Kim, Y. e Yang, S. (2014). A SDN-oriented DDoS blocking scheme for botnet-based attacks. Em *IEEE International Conference on Ubiquitous and Future Networks (ICUFN)*, p. 63–68.

Ltd., R. (2020). Ddos attack definitions - ddospedia. https://security.radware.com/ddos-knowledge-center/ddospedia/http-challenge/.

Mininet (2019). Mininet. https://github.com/mininet/mininet/wiki/documentation.

NETSCOUT (2020). What is ddos. https://www.netscout.com/what-is-ddos.

Nngroup (2019). Powers of 10: Time scales in user experience. https://www.nngroup.com/articles/powers-of-10-time-scales-in-ux.

Redis (2019). Redis 3.0.6. https://redis.io/.

Ryu (2019). Ryu. https://osrg.github.io/ryu/.

VirtualBox (2019). Virtualbox. https://www.virtualbox.org.

Wang, H., Xu, L. e Gu, G. (2015). Floodguard: A DoS attack prevention extension in software-defined networks. Em *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, p. 239–250.