UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

AS-DEFENDER: USANDO SDN PARA COMBATER O SEQUESTRO DE PREFIXOS IP NA INTERNET

Marcio Vinicius de Queiroz Santos

Orientador

Sidney Cunha de Lucena

RIO DE JANEIRO, RJ - BRASIL
Setembro de 2018

AS-DEFENDER: USANDO SDN PARA COMBATER O SEQUESTRO DE PREFIXOS IP NA INTERNET

Marcio Vinicius de Queiroz Santos

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO). APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

Aprovada por:

Sidney Cunha de Lucena, D.Sc. - UNIRIO

Carlos Eduardo Ribeiro de Mello, D.Sc. - UNIRIO

José Ferreira de Rezende, Dr. - UFRJ

Magnos Martinello, Dr. - UFES

RIO DE JANEIRO, RJ - BRASIL
Setembro de 2018

Santos, Marcio Vinicius de Queiroz.
S586 AS-Defender: Usando SDN para combater o sequestro de prefixos IP na Internet / Marcio Vinicius de Queiroz Santos. - Rio de Janeiro, 2018.
128 f.; 30 cm

Orientador: Lucena, Sidney Cunha de.
Dissertação (Mestrado) - Universidade Federal do Estado do Rio de Janeiro, Programa de Pós-Graduação em Informática, 2018.

1. BGP. 2. Sequestro de prefixo IP. 3. SDN. 4. OpenFlow. 5. Segurança. I. Lucena, Sidney Cunha de, orient. II. Universidade Federal do Estado do Rio de Janeiro. Centro de Ciências Exatas e Tecnologia. Curso de Mestrado em Informática. III. Título.

CDD – 007.33

Não poderia atribuir a apenas uma pessoa. À minha mãe, Ana, que crê no meu potencial até mais do que eu mesmo. À minha avó, Marilena, que nunca mediu esforços para me dar a melhor educação possível. À minha tia, Luciana, que sempre cuidou de mim como se fosse um filho. À minha irmã, Flávia, que é minha parceira desde que nasci. À minha esposa, Rychelle, que eu amo muito e me impulsiona a cada vez mais ser um homem melhor.

Agradecimentos

Primeiramente eu gostaria de agradecer a toda a minha família, especialmente à minha mãe Ana e à minha avó Marilena, além da minha tia Luciana, minha irmã Flávia e minha esposa Rychelle. Eu agradeço o apoio e as frases de otimismo, sempre me dizendo que tudo daria dar certo. Não posso esquecer da compreensão nos momentos em que fiquei ausente, e vocês, sabendo do motivo, só me perguntavam como os estudos estavam indo.

Agradeço novamente à minha esposa, melhor amiga e, em breve, PhD Rychelle, por sempre(foram muitas vezes!) me colocar para cima, me ajudar nos momentos em que eu tive que ficar dedicado aos estudos e inclusive me dar dicas para superar os desafios do mestrado. Desde que você entrou na minha vida, venho me tornando mais disciplinado e seguindo o seus exemplos. Você é o meu alicerce e eu não teria conseguido nada disso sem você. Eu amo você!

Não posso me esquecer dos amigos da pesquisa! Muito obrigado Vinicius, Wilson, Thiago Saraiva, Pedro, Helio e André! Você têm uma parcela muito grande nessa conquista. Como não lembrar daquela turma de Modelagem de Sistemas de Computação e Comunicação, além de uma ótima disciplina, foi o inicio de uma amizade que rendeu muitos momentos bacanas, ajudas nos estudos e até mesmo desabafos(Todo mundo!) nos momentos de dificuldade.

Ainda agradeço aos meus colegas do Laboratório de Distribuição e Redes pelo convívio e aprendizado semanal. Foram muitas reuniões e debates que certamente me fizeram crescer como profissional e ser humano.

Sou extremamente grato ao meu orientador Sidney! Lembro-me como se fosse ontem da nossa conversa na sala dos professores para falar sobre o início das pesquisas. Muito obrigado por me ajudar nesta caminhada, pelas conversas e por confiar na pesquisa, sempre tendo paciência e estando disponível para me ajudar a superar

os desafios.

Agradeço sinceramente à Universidade Federal do Estado do Rio de Janeiro juntamente com todos os seus funcionários, pela oportunidade de adquirir conhecimento em uma instituição de ensino superior de alta qualidade. Certamente as aulas que recebi, bem como os serviços que foram prestados foram relevantes para a minha formação acadêmica.

Aos meus companheiros de trabalho Álvaro e Carlos Mota fica um imenso agradecimento e uma grande dívida também, por terem gasto seu tempo me ajudando em vários momentos, como se todos os meus pedidos fossem urgentes.

Ainda em relação ao trabalho, eu agradeço a toda a equipe do suporte, em especial ao meu chefe Ricardo Barcelos, pelo apoio e compreensão com as necessidades de estudo.

Agradeço também à CPRM como instituição, meu emprego, por me apoiar nessa empreitada.

Aos membros da banca, professores Carlos Eduardo, Magnos e Rezende, agradeço por vocês aceitarem o convite para avaliar o meu trabalho. Para mim isso é uma honra e ao mesmo tempo um aprendizado.

Deixo também um agradecimento para aquele que se propõe a ler este trabalho. Espero que ele seja de grande valia e contribua para os seus objetivos. Ademais, estou disponível para quaisquer dúvidas ou comentários sobre a pesquisa.

E finalmente agradeço àquele que esteve ao meu lado a todo momentos, embora não visível aos meus olhos mas sim no meu pensamento. Deus, eu agradeço pelo auxilio diário e por todos os sinais e situações que você me fez passar para me mostrar que eu estava no caminho certo e que era possível chegar lá. Também peço desculpas por, em alguns momento, não ter confiado nisso. Agradeço também a todos que me ajudaram a partir do outro plano sem que eu tivesse ciência.

Santos, Marcio Vinicius de Queiroz. **AS-Defender: Usando SDN para combater o sequestro de prefixos IP na Internet**. UNIRIO, 2018. 99 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO.

RESUMO

O protocolo BGP(*Border Gateway Protocol*) desempenha um papel fundamental no roteamento global. Ele tem como função determinar os caminhos que os dados devem percorrer entre os sistemas autônomos na Internet. Infelizmente, apesar da sua importância, o BGP foi concebido sem qualquer mecanismo de segurança que permita garantir a autenticidade de um anúncio de rota e, diante disso, atacantes podem sequestrar um tráfego de interesse mediante o anúncio de rotas falsas, realizando o ataque conhecido como sequestro de prefixo. Atualmente, as redes definidas por *software* e o protocolo *OpenFlow* têm conferido um grande poder de inovação aos administradores de rede, permitindo a programabilidade do comportamento da rede, além do seu controle através de um ponto logicamente centralizado. Analisando essas propriedades, constatou-se a oportunidade de combinar os benefícios do SDN visando a criação de um recurso para combater o sequestro de prefixos IP, por meio de técnicas de detecção e mitigação existentes. Esta pesquisa apresenta o *AS-Defender*, um sistema autônomo baseado em SDN que visa combater o sequestro de prefixos IP, detectando e mitigando ataques direcionados aos prefixos dos seus vizinhos. Como prova de conceito, foi criado um protótipo utilizando o controlador *Ryu*, que foi submetido a experimentos onde ataques foram executados em dois cenários diferentes. O primeiro cenário teve o objetivo de mostrar como o mecanismo proposto funciona, enquanto o segundo experimentou a solução sob diferentes circunstâncias. Os resultados mostraram a eficácia do uso do *AS-Defender* no combate ao ataque de sequestro de prefixo e como as diferentes variáveis às quais um ambiente de comunicação inter-domínios está submetido podem influenciar na eficiência dessa solução.

Palavras-chave: BGP, Sequestro de prefixo IP, SDN, OpenFlow, Segurança.

ABSTRACT

The BGP (Border Gateway Protocol) has a fundamental role in global routing. Its goal is to determine paths inwhich data may flow from one autonomous system to another in the Internet. Despite its importance, unfortunatelly BGP was conceived without any security mechanism to guarantee the authenticity of a route prefix announcement. As a result, attackers can hijack traffic of their interest by announcing false routes, which implies in a kind of attack known as prefix hijacking. Nowadays, software-defined networking and OpenFlow protocol have conferred a great power of innovation to network administrators, allowing for the programmability of network behavior and for its control through a logically centralized spot. Analyzing these properties, we verified an opportunity to use the benefits of SDN paradigm in order to create a resource to combat prefix hijacking, using existing detection and mitigation techniques. This research presents AS-Defender, a SDN-based autonomous system that aims to combat IP prefix hijacking by detecting and mitigating attacks targeted to the prefixes of its neighbor ASes. As a proof of concept, a prototype was created using the Ryu controller, which was submitted to experiments where attacks were performed in two different scenarios. The first scenario had the objective of showing that the proposed mechanism works properly, while the second one investigated the solution under different circumstances. The results attested the efficacy of AS-Defender in combating prefix hijacking and showed how different circunstances of the inter-domain environment can influence the efficiency of the proposed solution.

Keywords: BGP, prefix hijacking, SDN, OpenFlow, Security.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Nomenclaturas

BGP	*Border Gateway Protocol*
AS	*Autonomous System*
ASN	*Autonomous System Number*
IP	*Internet Protocol*
TCP	*Transmission Control Protocol*
iBGP	*Internal Border Gateway Protocol*
eBGP	*External Border Gateway Protocol*
RFC	*Request for Comments*
OSPF	*Open Shortest Path First*
RIP	*Routing Information Protocol*
MED	*Multi Exit Discriminator*
IGP	*Interior Gateway Protocol*
EIGRP	*Enhanced Interior Gateway Routing Protocol*
MITM	*Man In The Middle*
RIB	*Routing Information Base*
RTT	*Round Trip Time*
TTL	*Time To Live*
MOAS	*Multiple Origin Autonomous System*
SDN	*Software Defined Networks*
API	*Application Programing Interface*
QoS	*Quality Of Service*
TLS	*Transport Layer Security*
REST	*Representational State Transfer*
YAML	*YAML Ain't Markup Language*
ICMP	*Internet Control Message Protocol*
JSON	*JavaScript Object Notation*
BIRD	*BIRD Internet Routing Daemon*
VLAN	*Virtual Local Area Network*
EGP	*Exterior Gateway Protocol*
ISP	*Internet Service Provider*

1. Introdução

A Internet é um conjunto descentralizado de redes, cada uma controlada por um ou mais operadores sob uma política clara e definida. Tais redes, chamadas de Sistemas Autônomos (AS), são compostas tanto de *hosts* (que originam e recebem pacotes e são identificados por um IP) quando de equipamentos de comutação de dados (roteadores) que são responsáveis por encaminhar os dados. O conjunto desses roteadores formam o sistema de roteamento, que é responsável por propagar as informações sobre o caminho para cada rede e tomar decisões de roteamento consistentes para transmitir um pacote da sua origem ao destino [2].

Nos primórdios da Internet, quando esta era apenas restrita a algumas empresas, a segurança não era o foco e seus primeiros projetistas não pensaram em mecanismos que pudessem garantir a confiabilidade das comunicações. Diante disso, atualmente o sistema de roteamento global ainda permanece suscetível a ataques [7].

O protocolo de roteamento atualmente utilizado na Internet é o BGP (*Border Gateway Protocol*). A sua primeira versão foi desenvolvida no ano de 1989 e, desde então, outras versões foram implementadas até o, atualmente utilizado, BGPv4 [8]. Como um algoritmo de roteamento inter-domínios, o BGP é responsável por propagar as informações de alcançabilidade das redes e conectar os ASes na Internet. Essa troca de mensagens é feita através mensagens de anúncios de prefixos, que são trocadas entre os roteadores. O BGP não verifica a validade de um anúncio de prefixo, uma vez que não há mecanismos para garantir a autenticidade da origem de um anúncio. Assim sendo, um atacante com o controle de um roteador na rede BGP poderia facilmente realizar anúncios de prefixos que não são da sua propriedade e redirecionar uma parte do tráfego da Internet para a realização de atividades criminosas [9]. Esse tipo de ataque é conhecido como sequestro de prefixo (ou *prefix hijacking*).

O *prefix hijacking* nem sempre ocorre em função de uma atividade maliciosa. Uma vez que ele é causado por um anúncio de um prefixo que não é de propriedade do AS anunciante, em alguns casos uma configuração incorreta de um roteador BGP pode desencadear o problema [10]. Em alguns casos, *o prefix hijacking* pode ser considerado um tipo especial de ataque de Negação de Serviço (DOS), pois ele faz com que dados não cheguem mais na rede de destino e sejam desviados para uma outra direção [11]. Essa peculiaridade torna a sua identificação difícil, ao se confundir o ataque com outros tipos de interrupções [12].

Muitos trabalhos têm procurado encontrar formas de identificar o *prefix hijacking*, porém a maioria das propostas existentes [13, 14, 15, 16, 17] requerem alterações no próprio protocolo, em configurações do roteador ou até mesmo necessitam da criação de infraestruturas adicionais, como a criação de uma RPKI (*Resource Public Key Infrastructure*) [18]. Além disso, após a identificação do ataque, contramedidas se fazem necessárias, pois apenas identificar o ataque não impede que ele ocorra ou mesmo atenua os seus resultados, e, nesse aspecto, apesar de já existirem trabalhos que realizam a mitigação [19, 20], não existem ainda soluções de mitigação amplamente implantadas [7].

Nos últimos anos, as infraestruturas de dados públicos para monitoramento global do BGP (como o *RouteViews* [21] e o RIPE RIS [22]) têm melhorado muito e vêm se tornando um recurso valioso para o monitoramento do plano de controle BGP. Essa evolução abre uma possibilidade para obtermos uma visão mais abrangente dos dados que estão sendo trocados na Internet e não necessitarmos que o anúncio chegue a um AS específico para que o ataque seja identificado.

1.1 Motivação da pesquisa

O *prefix hijacking* permanece sendo motivo de preocupação para os operadores de rede e ainda causa grandes problemas no roteamento global. Nos últimos anos, foi possível verificarmos diversos eventos provenientes de ataques que geraram prejuízos para diversos domínios conhecidos.

Como exemplo, pode-se citar o caso do dia 24 de fevereiro de 2008, que foi marcado pelo sequestro do prefixo de rede 208.65.153.0/24, de propriedade do *YouTube*, pelo ISP *Pakistan Telecom*, o que resultou em um desvio no roteamento do tráfego em escala global que bloqueou o acesso ao *YouTube* de diversas partes da Internet [23].

Um outro notável ataque ocorreu entre fevereiro e maio de 2014 e foi descoberto por pesquisadores da *Dell SecureWorks*. Durante mais de dois meses, tráfegos em direção a *pools* de mineração de criptomoedas foram sequestrados, comprometendo mais de 51 redes em 19 ISP's (Internet Service Providers). O ataque gerou uma perda de mais de 9 mil dólares diários e 83 mil dólares durante todo o período [24].

Embora o BGP seja fundamental para o funcionamento da comunicação entre computadores em todo o mundo, ele não possui um mecanismo para autenticar a origem de um anúncio e, apesar de muitas pesquisas terem sugerido soluções para o *prefix hijacking* nos últimos anos, é improvável que alguma implantação completa ou em larga escala ocorra [25]. Diante da dificuldade de se corrigir essa fragilidade do BGP ou, até mesmo, substituí-lo em toda a Internet, é fundamental continuarmos a busca por soluções ou, ao menos, formas de reduzir os prejuízos decorrentes do ataque.

1.2 Objetivos

O presente trabalho tem como principal objetivo fornecer uma forma de combater o sequestro de prefixos IP, a partir de um sistema autônomo que realize a detecção e a mitigação do ataque direcionado aos seus vizinhos. Para isso, foi construído um protótipo de aplicação SDN, chamado de *AS-Defender*, fazendo uso da linguagem de programação Python e do controlador Ryu [26], que é um *framework* para programação em redes definidas por *software* que fornece diversos componentes mediante uma API, tornando possível a criação de aplicações para controlar e gerenciar uma rede.

A hipótese que se deseja investigar é se a possibilidade de programar totalmente o comportamento de uma rede, mediante o uso dos conceitos do paradigma SDN, pode auxiliar na realização de medidas contra o sequestro de prefixo. Além disso, uma outra questão que este trabalho busca entender é se um AS pode tirar proveito da sua posição estratégica em uma topologia para realizar algumas medidas visando mitigar o ataque, como por exemplo, mudar a direção do tráfego.

O *Ryu* foi escolhido como controlador, dentre outros motivos, por já possuir recursos para a implantação de um processo BGP, e portanto possibilitar a instanciação de um sistema autônomo que se comunique de acordo com os padrões estabelecidos na RFC do protocolo. Através de sua API, é possível criar um componente que possa invocar um processo BGP e, ao mesmo tempo, monitorar os ataques

direcionados a outros ASes, tomando ações conforme necessário.

É importante ressaltar que este trabalho não pretende criar um novo algoritmo de detecção ou mitigação do ataque, mas sim utilizar mecanismos e conhecimentos previamente descritos em outras pesquisas, que combinados à arquitetura proposta, baseada no paradigma SDN, possam cumprir o objetivo citado.

1.3 Contribuições

A principal contribuição deste trabalho é a apresentação de um mecanismo de detecção e mitigação do sequestro de prefixos IP, que combina recursos existentes, através de uma arquitetura SDN, formando um AS capaz de proteger os seus vizinhos. Além disso, um tipo de mitigação é mostrado e avaliado, para entendermos se simplesmente o fato do *AS-Defender* vir a estar diretamente conectado à maioria dos demais ASes torna viável a mitigação através do redirecionamento do tráfego para as vítimas.

Embora tenha sido desenvolvido especificamente na plataforma Ryu, uma contribuição implícita são os conhecimentos disponibilizados sobre os componentes que foram utilizados no trabalho e que possibilitam que o *AS-Defender* seja reproduzido para funcionar acoplado a outro controlador.

Para o gerenciamento dos dados do plano de controle referentes ao *switch*, foi construído um módulo separado, que pode ser consumido por outras aplicações SDN que desejem realizar operações semelhantes. Obviamente, para usos em outros casos, provavelmente algumas mudanças no módulo serão necessárias.

Uma outra contribuição são os conhecimentos fornecidos à partir dos resultados, que mostram que, além dos mecanismos impostos pelo protótipo, políticas baseadas em filtros BGP, assim como as conexões entre os ASes, podem influenciar no processo de mitigação.

Dentre as contribuições fornecidas pelo projeto *AS-Defender*, a mitigação possui uma relevância maior, dado que poucos trabalhos ainda exploram esse aspecto. Portanto, esta pesquisa tem como principal colaboração as formas de mitigação do ataque de *prefix hijacking* através do uso das redes definidas por *software*.

1.4 Estrutura da dissertação

Após este capítulo introdutório e motivacional, procedemos com a seguinte composição textual:

- O Capítulo 2 aborda todo o conhecimento teórico sobre o protocolo BGP, necessário para a compreensão da proposta, da sua execução e da validação dos resultados.
- O Capítulo 3 fornece a teoria sobre as redes definidas por *software*, um outro componente importante para o entendimento do trabalho, finalizando assim a fundamentação teórica.
- O Capítulo 4 é dedicado para a descrição da proposta sob uma abordagem *top-down*, desde os aspectos gerais, passando pelas fases de funcionamento e os processos de detecção e mitigação do ataque, até uma breve consideração sobre os trabalhos relacionados.
- O Capítulo 5 esclarece os detalhes da implantação do protótipo em um aspecto prático, abordando o ambiente computacional, os componentes utilizados e os desafios encontrados.
- O Capítulo 6 valida a proposta através dos cenários experimentais e da metodologia utilizada, além de discutir os resultados obtidos a partir dos ataques realizados e abordar as limitações identificadas.
- O Capitulo 7 encerra este trabalho, apontando as perspectivas futuras e finalizando com considerações a respeito do cenário dos estudos sobre o campo de atuação.

2. O BGP(*Border Gateway Protocol*)

Este capítulo fundamenta os conceitos do protocolo BGP, além de descrever o ataque *prefix hijacking*, seus tipos e suas formas de detecção e mitigação.

2.1 Introdução ao BGP

O BGP é um protocolo de roteamento interdomínios (entre sistemas autônomos) que possibilita a comunicação de dados através de toda a Internet. Em sua essência, o BGP é um protocolo absolutamente crítico para o funcionamento da Internet uma vez que ele é o responsável por agregar todas as suas redes.

No BGP, pares de roteadores trocam as informações de roteamento através de uma conexão utilizando o protocolo de transporte TCP na porta 179. A conexão BGP ocorre em cada enlace que liga diretamente dois roteadores, no qual eles trocam informações de roteamento. Os roteadores nas extremidades da conexão são conhecidos como pares BGP e tanto a conexão quanto as mensagens enviadas através dela constituem o que chamamos de sessão BGP [1].

Uma sessão BGP pode abranger tanto roteadores de um mesmo sistema autônomo(AS) como também de dois ASes diferentes. Caso na sessão estejam envolvidos dois roteadores BGP do mesmo sistema autônomo, ela recebe o nome de sessão BGP interna (iBGP), enquanto que com dois roteadores de borda, em sistemas autônomos diferentes, nós a chamamos de sessão BGP externa (eBGP).

O BGP permite que cada AS saiba quais redes podem ser alcançadas por meio dos seus ASes vizinhos. Essas redes são prefixos IP anunciados entre os sistemas autônomos, para informar aos seus vizinhos que o caminho para uma determinada rede pode ser alcançado por um enlace.

Em uma comunicação, os roteadores trocam quatro tipos de mensagens, cada uma com a sua finalidade em uma sessão BGP.

A mensagem *OPEN* é transmitida entre os roteadores assim que uma conexão TCP é estabelecida, quando eles querem estabelecer uma sessão. Além disso, a *OPEN* é responsável pela troca de parâmetros entre os roteadores. Diferentemente de alguns protocolos de roteamento, os nós BGP não são descobertos automaticamente, mas configurados para se comunicarem com um par específico.

A mensagem *KEEPALIVE* é enviada periodicamente para garantir que a conexão entre os pares esteja sempre disponível. Se um dos roteadores ficar ausente por um determinado tempo, a conexão é encerrada.

A mensagem *NOTIFICATION* é enviada quando uma condição de erro inesperada é encontrada. Após enviar a mensagem, o roteador encerra imediatamente a sessão.

A mensagem *UPDATE* é utilizada para efetivamente transferir as informações de roteamento entre os pares. Tanto os anúncios de rotas alcançáveis quanto os caminhos que não são mais acessíveis são informados através de um *UPDATE*. Os *UPDATES* que contenham inclusões de rotas são conhecidos como anúncios, enquanto que para retiradas, costumam ser chamados de *WITHDRAW*.

Na Figura 2.1 é possível ver o funcionamento de uma topologia BGP, onde sistemas autônomos trocam informações para, de forma distribuída, ocorrer o roteamento de dados na Internet.

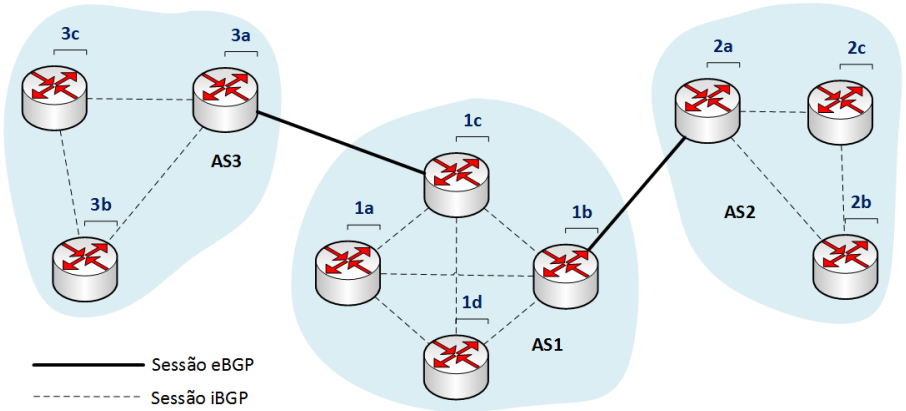


Figura 2.1: Funcionamento do BGP - Imagem adaptada de [1]

Cada conjunto de roteadores com o fundo sombreado representa um AS, onde os roteadores internos realizam uma sessão iBGP, enquanto cada equipamento de borda troca informações de roteamento através do eBGP.

Para que um pacote de dados possa sair de uma origem no *AS3* e chegar a um destino no *AS2*, é necessário que o *AS3* saiba que, para chegar ao *AS2*, o pacote deve ser entregue ao *AS1*. Essa informação de alcançabilidade é trocada através da mensagem *UPDATE*, que é propagada entre os sistemas autônomos conforme a política de cada um.

Nesse caso, o *AS2* deve anunciar, através de uma mensagem *UPDATE* ao *AS1*, que para alcançar um determinado prefixo de sua propriedade, os pacotes devem ser encaminhados para o seu roteador de borda *2a*. O *AS1* receberá tal anúncio e informará ao *AS3* que, para chegar ao prefixo anunciado, o pacote de dados deve ser encaminhado para o seu roteador *1c*, que por sua vez encaminhará ao roteador de borda *1b* para que o pacote chegue até o destino. Essa é uma explicação genérica sobre como os sistemas autônomos na Internet ficam sabendo sobre por onde encaminhar um pacote de dados para um determinado destino.

Falando um pouco mais a fundo sobre essa troca de mensagens de *UPDATE* entre os roteadores BGP, no exemplo da Figura 2.1, uma sequencia de fatos seguiria as seguintes etapas:

- O *AS2*, através de sua sessão eBGP entre o seu roteador de borda *2a* e o roteador de borda do *AS1*, *1b*, envia uma mensagem *UPDATE* informando a lista de prefixos que podem ser alcançados por *AS2*(incluindo os prefixos de sua propriedade)
- No *AS1*, o roteador de borda *1c*, através da sua sessão eBGP com o roteador *3a*, envia uma mensagem *UPDATE* com os prefixo de rede que podem ser alcançados através do *AS1*.

Da mesma forma, o fluxo de mensagens pode percorrer o caminho inverso, ou seja, o *AS3* pode enviar mensagens *UPDATE* com os prefixos de rede que são alcançáveis através dele para o *AS1*, e este propagar essa mensagem para o *AS2*.

Quando qualquer um dos ASes recebe anúncios de prefixos nos seus roteadores de borda essas informações são distribuídas entre os demais roteadores internos do AS através das suas sessões iBGP [1]. A definição sobre qual é o menor caminho interno até um roteador de borda em um AS, para que os dados seja roteados até um outro AS, é uma tarefa do algoritmo de roteamento intra-AS que está sendo utilizado, que pode ser por exemplo, o OSPF [27] ou RIP [28], que não são o foco deste trabalho.

2.2 O atributo *AS PATH*

O atributo *AS PATH* é um atributo obrigatório, ou seja, ele está presente em todas as mensagens *UPDATE* do BGP. O *AS PATH* é composto de uma lista com todos os ASN's (*Autonomous System Numbers*) de cada AS que devem ser atravessados para que a origem do anúncio seja alcançada [29].

Cada vez que um roteador envia uma mensagem *UPDATE* ao seu vizinho, ele acrescenta o seu próprio ASN ao lado esquerdo da lista de ASN's no *AS PATH* e conforme os anúncios de prefixos são propagados pela Internet essa lista vai crescendo formando o caminho e todos os saltos (cada salto corresponde a um AS) a serem percorridos até a origem. A composição do *AS PATH* pode ser vista na tabela 2.1:

65501	65502	65503	65504	65505

Tabela 2.1: Sequência de ASN's no atributo *AS PATH*

A mensagem *UPDATE* que contém o atributo *AS PATH* com a lista de ASNs descrita na tabela 2.1 indica que o *AS65505* é o proprietário do prefixo uma vez que foi o primeiro AS realizar o *UPDATE*, o *AS65504* foi o primeiro AS a receber tal anúncio e consequentemente pode ser entendido como um vizinho do *AS65505*. Continuando a analisar a lista vemos que os ASes *65503*, *65502* e *65501* são os demais ASes por onde o *UPDATE* passou respectivamente.

Ao anunciar um prefixo, o *AS PATH* é populado da direita para a esquerda e ,consequentemente, lendo a sequência de ASN´s no sentido natural, da esquerda para a direita, podemos verificar todo o caminho que um pacote percorreria caso fosse enviado pelo *AS65501* com destino ao prefixo anunciado pelo *AS65505* (origem do anúncio) desde que essa rota fosse a escolhida como melhor caminho para o BGP.

O propósito principal do atributo *AS PATH* é o de evitar *loops* de informação de roteamento, pois, caso esse atributo não existisse, o BGP se comportaria de forma semelhante ao protocolo RIP [28], que é um protocolo de vetor de distâncias, e assim estaria sujeito ao problema conhecido como contagem ao infinito [1].

Um outro importante aspecto *AS PATH* é que ele faz parte dos critérios de seleção da melhor rota no BGP. Dentre outros critérios, o BGP avalia o comprimento do *AS PATH* das rotas para um mesmo prefixo e da preferência à(s) rota(s) com o *AS PATH* mais curto, preferindo o caminho com um menor número de ASes entre a origem e o destino.

O fato do *AS PATH* ser um importante componente do processo de seleção de rotas do BGP faz com que ele seja alvo de manipulação por parte de atacantes, que tenham como objetivo influenciar no processo de decisão e no roteamento de dados na Internet para, a partir daí, capturar informações que não são de sua propriedade através do *prefix hijacking* [30].

2.3 Critérios de seleção da melhor rota no BGP

Quando um roteador recebe dois anúncios que contemplam o mesmo destino, é necessário que um deles seja escolhido como o melhor caminho e, para isso, o BGP possui diversos critérios de seleção que seguem uma ordem de avaliação sequencial, enquanto a melhor rota não for encontrada.

É formada uma lista com todos os caminhos válidos para um determinado prefixo, sendo o primeiro escolhido como o melhor caminho atual. A partir desse momento, o BGP compara esse caminho com o próximo da lista até que todos os caminhos sejam avaliados.

A implementação do BGP empregada neste trabalho é a mesma definida na documentação do BIRD [31], dado que este foi o *software* utilizado na validação experimental.

O primeiro critério de avaliação é o atributo de preferência local, onde o caminho com o maior valor é o escolhido. Por padrão, todas as rotas recebem um valor de preferência local igual a 100, ficando a critério do administrador configurar um valor diferente, caso deseje dar preferência a alguma rota. Este atributo não é passado entre ASes diferentes, de modo que ele é apenas configurado internamente em um AS.

Em seguida, é avaliada a largura do atributo *AS Path*, onde rotas com um numero menor de ASes entre a origem e o destino são preferidas em detrimento das demais.

O próximo critério analisado é o atributo *ORIGIN*, que indica a origem de um prefixo e pode assumir três valores: IGP, EGP ou *incomplete*. O primeiro valor significa que a origem do prefixo é um protocolo de roteamento interno, enquanto o segundo define que o prefixo foi originado pelo protocolo EGP [32], atualmente considerado obsoleto, enquanto *incomplete* denota que a rota foi aprendida a partir de outros meios. A ordem de preferência para esse atributo é IGP-EGP-*incomplete*.

Posteriormente, é escolhido o menor valor do atributo MED, que é usado para sugerir aos vizinhos sobre o caminho preferido, em um AS que possua mais de um ponto de entrada.

Caso ainda não tenha sido possível definir o caminho, o BGP recorre a mais três critério de decisão na respectiva sequencia: as rotas recebidas via eBGP, a menor distância interna para um roteador de borda e a rota recebida pelo roteador com o menor valor *router ID*, usualmente definido como um dos seus IP´s [33].

2.4 O ataque *prefix hijacking*

No BGP, não há uma forma de um roteador verificar a exatidão de um anúncio de prefixo, uma vez que nenhum mecanismo de autenticação de origem é implementado [9]. Sendo assim, é teoricamente fácil para um atacante, com acesso direto a um roteador na rede BGP, realizar o um anúncio falso.

Para um atacante redirecionar o fluxo de dados pertencente a um AS para um outro roteador, que esteja sob o seu controle, é necessário que ele consiga influenciar tal fluxo através de mensagens *UPDATE* falsas de modo que, todos os ASes, ou pelos menos alguns, prefiram rotear os dados para o caminho que ele deseja [34].

Essa influência realizada por um atacante, ou até mesmo uma má configuração de um roteador BGP, podem dar inicio ao *prefix hijacking*. Há diversos tipos e propósitos para a realização do *prefix hijacking*, porém, todos começam de uma mesma forma, um anúncio de prefixo por um AS que não possui a sua propriedade.

A seguir, na Figura 2.2, é representado o *prefix hijacking* e o seu funcionamento. Na imagem nós temos a presença de uma topologia BGP com os ASes enumerados de 1 a 6, onde cada roteador faz referência a um AS. O *AS1* é proprietário do prefixo 123.123.0.0/16 e envia uma mensagem *UPDATE* para informar o *AS2* que, para alcançar a rede 123.123.0.0/16, o pacote deve ser enviado para ele. O *AS2* recebe o *UPDATE*, atualiza a sua tabela de encaminhamento e anuncia para o seu vizinho *AS3* que, para alcançar o prefixo de propriedade do *AS1* o pacote deve ser entregue a ele. O *AS3* também atualiza a sua tabela de encaminhamento e anuncia a rota recebida para o *AS4*, até que a propagação do anúncio chegue ao *AS5*, que também realiza as respectivas atualizações.

Em seguida, o *AS6* envia um *UPDATE* com o subprefixo 123.123.0.0/24, que não é de sua propriedade, e o anúncio de rota se propaga até o *AS5*. Após o

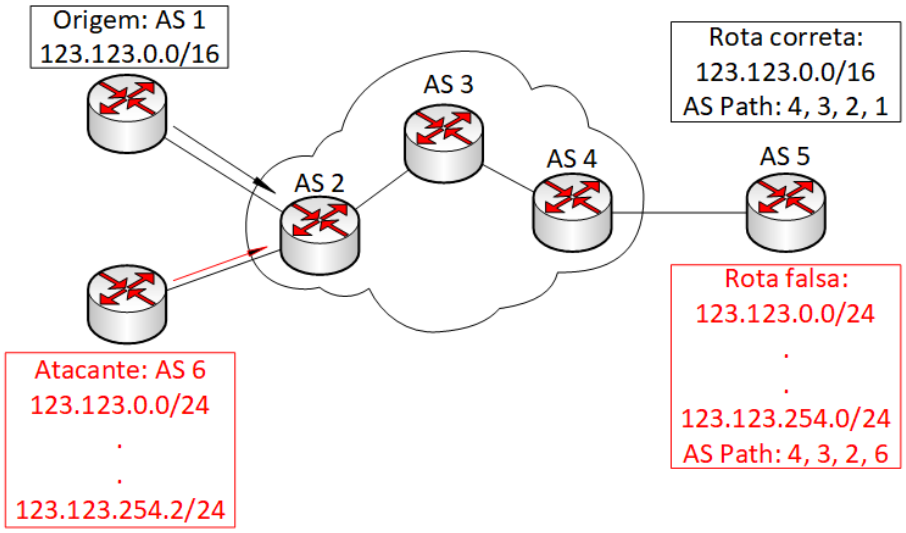


Figura 2.2: *Prefix hijacking* - Imagem adaptada de [2]

recebimento do anúncio falso, assim que o *AS5* enviar um pacote com destino a sub-rede anunciada pelo *AS6*, os dados percorrerão o caminho *AS4-AS3-AS2*. Ao receber o pacote, a tabela de encaminhamento do *AS2* terá uma informação semelhante a Tabela 2.2 :

Network	Next Hop	Metric	LocPrf	Weight	Path
123.123.0.0/16	192.168.1.1	0			1 i
123.123.0.0/24	192.168.6.1	0			6 i

Tabela 2.2: Exemplo de tabela de encaminhamento no BGP

A rota que será escolhida para a entrega dos dados é a que se conecta ao *AS6* pois, em sua tabela de encaminhamento, o prefixo mais específico que atende ao IP de destino tem como próximo salto o IP 192.168.6.1. Nesse momento o *AS6* sequestrou um subprefixo pertencente ao *AS1* e o roteamento de dados para os destinos inseridos no prefixo 123.123.0.0/24 está comprometido.

A situação descrita acima representa uma forma básica do ataque, porém, o *prefix hijacking* é dividido em subtipos e pode ser realizado por atacantes com vários propósitos, que vão desde realizar ataques de negação de serviço [35] até interceptar dados e inspecionar o seu conteúdo para obter informações sensíveis [36].

Os efeitos de um *prefix hijacking* podem "poluir" muitos ASes e assim causar um grande desvio no roteamento de dados. Alguns tipos de ataque são de melhor detecção enquanto outros são extremamente difíceis de observar. Felizmente, há ataques que não são comuns e exigem que o atacante tenha o acesso direto a um

roteador na rede, dificultando a sua ocorrência.

2.4.1 Tipos de *prefix hijacking*

O *prefix hijacking* pode ser classificados em alguns tipos. Essa classificação depende sob qual ponto de vista o ataque está sendo analisado. Este trabalho se baseou na classificação introduzida em outras pesquisas [20, 19, 37, 9] para avaliar os tipos de *prefix hijacking* e as suas consequências, utilizando a seguinte taxonomia de classificação:

Classificação pelo prefixo afetado

- **Sequestro de Prefixo**: O atacante anuncia exatamente o mesmo prefixo que a vítima e como, geralmente, o BGP prefere rotas com o menor *AS PATH*, a decisão da melhor rota dependerá da posição do atacante ou de outros critérios de decisão.
- **Sequestro de Subprefixo**: Nesse caso o atacante anuncia um subprefixo da vítima e, em função da preferência de roteamento para prefixos mais específicos, a Internet inteira pode ser populada com um anúncio desse tipo.
- ***Squatting***: Em alguns casos, um AS possui a propriedade de prefixos que ainda não são anunciados, pois os IP´s referentes a esses prefixos ainda não estão sendo usados. O *Squatting* [38] se caracteriza pelo sequestro desses prefixos que, devido a sua não utilização, podem popular toda a Internet sem que o proprietário perceba.

Classificação pelo *AS PATH* anunciado

- ***ORIGIN hijacking* (Sequestro de Origem ou Tipo 0)**: O atacante anuncia como seu um prefixo que não possui. Nesse caso, ao verificar o *AS PATH*, a posição do proprietário (valor mais a direita) aparece diferente do esperado.
- ***Type-N* (N ≥ 1) *hijacking* (Sequestro Tipo N)**: O atacante deliberadamente anuncia um caminho falso para um prefixo que ele não possui. O *AS PATH* contém o ASN do atacante como último salto enquanto a sequencia de ASes não representa um caminho valido. A posição do link falso representa o valor de N. Por exemplo, sendo o atacante o *AS2* em um *AS PATH* com

o valores [*AS2*, *AS1*] teremos o Tipo 1, pois o *AS2* está na posição 1 do *AS PATH* (inicia a partir da posição 0 que é o valor mais a direita) e não possui um link real com o *AS1*.

Classificação pela manipulação do tráfego

- ***Blackholing*:** O atacante anúncia uma rota falsa e, ao receber o fluxo de dados, descarta todos os pacotes com direção ao IP da vítima. O anúncio pode ser enviado com um endereço IP inválido e, consequentemente, quando os dados forem encaminhados para o IP anunciado não chegarão a lugar algum. O *Blackholing* gera um ataque de negação de serviço que pode interromper o acesso à um recurso disponibilizado pela vítima.
- ***Imposture*:** O atacante realiza um anúncio de rota falso e configura uma máquina que se comporta exatamente igual a da vítima. Essa manipulação de tráfego pode possibilitar ao atacante a realização de um ataque de *phishing* de uma forma praticamente imperceptível pela vítima, que acessa um serviço com o mesmo nome, IP e comportamento que o real.
- **Interceptação**: O atacante anuncia uma rota falsa para uma máquina que, posteriormente, encaminha tráfego ao servidor da vítima. Isso torna possível a realização de um ataque do tipo MITM (*Man-In-The-Middle*). Esse tipo de ataque é muito difícil de ser detectado uma vez que a vítima não reconhece nenhuma mudança com relação a chegada de fluxo de dados em seu AS.

2.4.2 Métodos de detecção do *prefix hijacking*

Detectar o *prefix hijacking* não costuma ser uma tarefa usual e em determinados tipos de ataque essa identificação até o momento é impossível. A grande proporção que a nossa Internet tomou, a falta de uma autoridade administrativa central, o fato de cada AS ter a sua política de roteamento e anúncio das redes, e até mesmo a característica intrínseca da topologia de roteamento interdomínio com diversas conexões entre os ASes, dificultam ainda mais a detecção do ataque [9].

Diversos trabalhos sugerem as mais diferentes abordagens visando a identificação do ataque, porém as formas de detecção mais comuns têm focado em alguns aspectos, como a análise dos dados do plano de controle [39, 40, 41, 20, 42, 43], do plano de dados [11, 18], da combinação entre ambos [12] e também através da correlação das informações obtidas [44].

As abordagens que envolvem a análise dos dados do plano de controle visam monitorar as informações de roteamento do BGP. Essas informações incluem anúncios BGP trocados entre os roteadores e até mesmo informações das suas tabelas de roteamento.

Como exemplos de serviços que fornecem esses dados, podemos citar o *Route Views* [21], RIPE [22] e o *BGPMon* [40, 45]. Esses serviços possuem roteadores que realizam sessões eBGP com diversos sistemas autônomos em vários lugares no mundo e, seus equipamentos, atuam como *Route Collectors*, que recebem todos os anúncios (mas, em geral, não anunciam prefixos próprios) e disponibilizam publicamente as informações em diversos formatos.

A detecção do *prefix hijacking* através do uso das informações do plano de dados é feita através da verificação contínua da Internet para identificar qualquer mudança no caminho dos dados [44]. Essa verificação, geralmente, envolve um grande número de redes e os caminhos dos dados são coletados por ferramentas como, por exemplo, o *traceroute*. Além disso, as informações do plano de dados, possibilitam a realização de outras técnicas para a detecção do *prefix hijacking* como, por exemplo, a contagem de saltos até um prefixo [11], verificação de diferenças no RTT/TTL [46] e até algumas métricas realizadas através da ferramenta *ping* [47, 44].

Na detecção através da combinação das informações do plano de controle com o plano de dados são utilizadas análises, tanto dos dados de roteamento BGP e de conflitos de MOAS, quanto das métricas de performance da rede através de *ping* e *traceroute*, onde ambas as informações são relevantes na identificação do ataque.

Na correlação [44] são utilizadas diversas fontes de detecção, como dados públicos de roteamento, histórico de *traceroute*, análise de IPs e informações de ASes vizinhos, onde todos esses dados são correlacionados com o objetivo de identificar uma anomalia em um dos prefixos, através de padrões de comportamento e, consequentemente, alertar sobre o *prefix hijacking*.

2.4.3 A mitigação do *prefix hijacking*

Em geral, uma vez que o *prefix hijacking* é identificado pelo operador de rede, este procura uma forma de buscar contato com o AS que está sequestrando o seu prefixo. Além disso, procura, manualmente, configurar os seus roteadores BGP para anunciar algumas das suas sub-redes e, assim, através da regra de roteamento por prefixo mais específico, conseguir atrair uma parte do tráfego na Internet.

Por exemplo, no caso de um ataque ao prefixo 10.20.0.0/23, o seu proprietário poderia anunciar as sub-redes 10.20.0.0/24 e 10.20.1.0/24. Ao anunciar essas duas sub-redes, desde que o ataque tenha sido realizado apenas no prefixo principal, todo o tráfego com destino aos seus dois sub-prefixos será novamente encaminhado ao proprietário [20]. A mitigação manual do ataque, em alguns casos, não atende às necessidades dos ASes que precisam se recuperar rapidamente de uma situação adversa [10].

Embora poucas, algumas soluções [20, 19] também realizam a mitigação automática, onde a mesma ação que seria tomada pelo operador, é feita por um sistema após a identificação de um ataque.

No caso de um sequestro de subprefixo, a mitigação torna-se mais difícil pois, ao anunciar o mesmo subprefixo, não há garantias de que todos os ASes preferirão encaminhar os dados para o proprietário. Em geral, ataques em prefixos /24 são bastante difíceis de mitigar através da estratégia de anúncios, porque prefixos /25 em diante costumam ser filtrados por roteadores. Logo, a eficácia desse mecanismo de defesa é limitada [20].

Mitigar o *prefix hijacking* vai além de influenciar o roteamento interdomínios através de anúncios e demanda outras abordagens afim de verificarmos a sua eficácia. A realização de anúncios BGP por um AS terceiro pode ser uma opção e, ao atrair o tráfego, este poderia entregar os dados novamente para a vítima [48]. Conforme a localização do AS que está anunciando o prefixo, isso pode ser um fator alavancador para a eficiência do mecanismo e, portanto, potencializar a quantidade de ASes que encaminharão os dados ao proprietário, ao invés do atacante.

2.5 Síntese

Neste capítulo foram abordados os aspectos conceituais do protocolo BGP, bem como o ataque *prefix hjacking* e as suas formas de detecção e mitigação. O entendimento dos conceitos aqui tratados é de grande importância para a compreensão tanto da proposta quanto da validação experimental.

3. As redes definidas por *software* e o protocolo *OpenFlow*

Este capítulo disserta sobre as redes definidas por software e o protocolo *OpenFlow*, iniciando por uma visão geral e em seguida conceituando cada componente da pilha de tecnologias envolvidas.

3.1 O conceito de SDN

Na Internet, toda a comunicação de dados e as interligações entre as redes são feitas através de *switches* e roteadores físicos, que realizam a comutação desses pacotes entre a origem e o destino. Tais equipamentos, geralmente, possuem uma interface (gráfica ou por linha de comando) pela qual o operador pode configurá-los e gerenciá-los. Essa interface provê um acesso a todos os recursos disponíveis nos dispositivos, porém, frequentemente escondem do operador os detalhes internos de realização dessas operações. Isso não costuma ser um problema, quando o que se deseja é utilizar uma funcionalidade já existente no equipamento.

Quando o operador precisa utilizar uma nova funcionalidade ou até mesmo um protocolo, só será possível utilizá-lo se o *firmware* do equipamento for compatível com o recurso que se deseja, pois, do contrário, será necessário fazer uma requisição de atualização ao fabricante. Essa atualização pode demorar ou até mesmo nunca ser disponibilizada pelo fabricante, impossibilitando o uso das novas tecnologias e demandando a compra de um novo equipamento, geralmente a um alto custo, para atender às novas necessidades. Dessa forma, isso acaba representando uma barreira à inovação nas redes e, como resultado, os administradores de rede ficam limitados a um conjunto pré-definido de comandos, que são disponibilizados pelo equipamento que utilizam.

Diante disso, a atual infraestrutura da Internet, centrada em *hardware*, sofre com diversas deficiências, como problemas de gerenciabilidade, flexibilidade e inovação [49]. Para superar essas limitações, o conceito de SDN foi proposto. A sigla SDN significa (em inglês) redes definidas por *software*, uma arquitetura de redes que tem como objetivo possibilitar a inovação através da programabilidade do comportamento das redes de computadores. Ela possui as seguintes características [50]:

- **Diretamente programável:** O controle da rede é diretamente programável através do uso de API's.
- **Ágil:** Possibilita administradores ajustarem o fluxo de tráfego em toda a rede conforme necessário.
- **Gerenciamento centralizado:** A inteligência da rede é logicamente centralizada e mantém uma visão global dos componentes.
- **Configurada através de programação:** Toda a rede pode ser configurada e gerenciada através de aplicações SDN que automatizam as suas necessidades.
- **Baseada em padrões abertos:** Simplifica o projeto da rede, uma vez que não dependente de nenhum dispositivo ou protocolo proprietário.

O aspecto fundamental do SDN é a separação entre o plano de controle e o plano de dados da rede. A funcionalidade de encaminhamento dos dados, incluindo as tabelas que são utilizadas por um equipamento para saber como lidar com a entrada de pacotes, fazem parte do plano de dados. Além disso, ao receber os pacotes, o plano de dados pode realizar diversas ações como por exemplo descartar, encaminhar e replicar os dados.

A lógica e os algoritmos que são usados para programar todo o comportamento do plano de dados reside no plano de controle. O plano de controle representa o cérebro, ou seja, a inteligência de um dispositivo de encaminhamento e portanto determina como as tabelas de encaminhamento e o plano de dados devem ser programados e configurados.

Nas redes tradicionais, cada dispositivo de rede possui o seu próprio plano de controle e de dados, onde o plano de controle tem como tarefa primária executar o protocolo de roteamento, enquanto que no plano de dados são realizadas as funções de encaminhamento.

No SDN, o plano de controle é desacoplado de cada um dos dispositivos de encaminhamento para um componente logicamente centralizado chamado controlador. A Figura 3.1 representa uma visão conceitual e todos os componentes básicos presentes em uma arquitetura SDN que são: os *switches* SDN, o controlador e as aplicações SDN, além das *API´s* que realizam a interface entre os componentes.

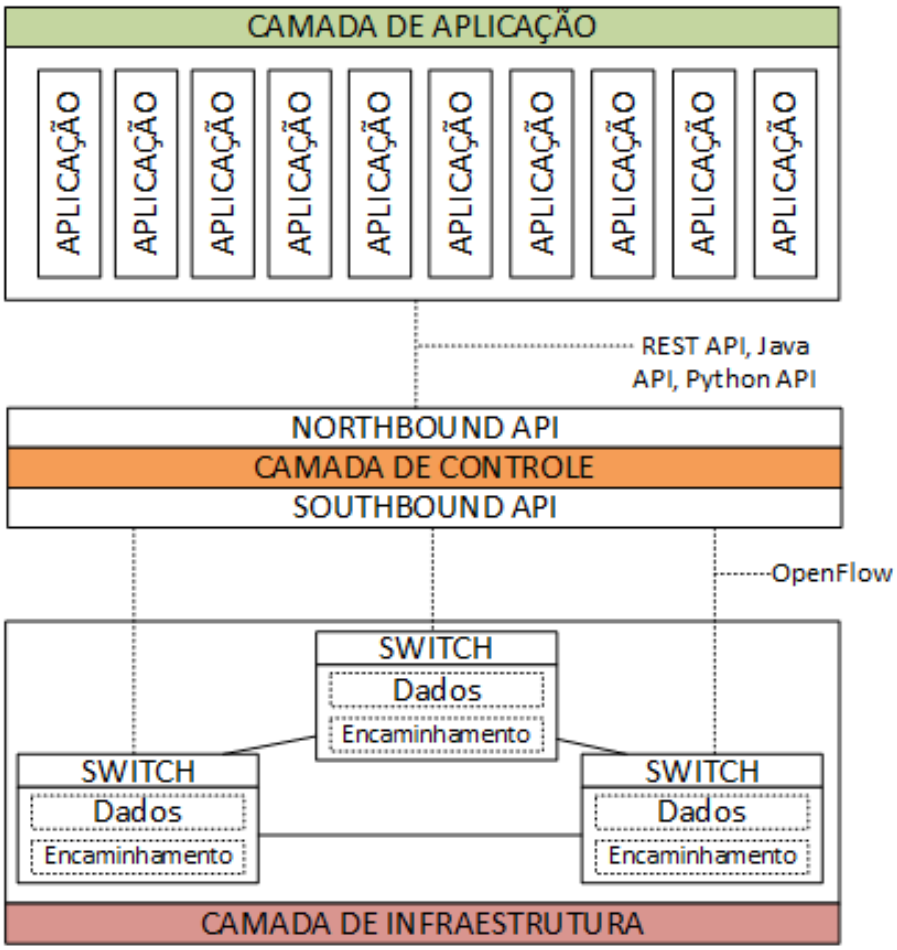


Figura 3.1: Arquitetura do SDN - Imagem adaptada de [3]

Os *switches* SDN são elementos de rede responsáveis por encaminhar os dados e realizar as funções do plano de dados e portanto possuem a funcionalidade de decidir o que fazer quando receberem um pacote em uma de suas portas. Essa decisão é tomada através da consulta a sua tabela de fluxos que contém as entradas de fluxos e as ações que devem ser realizadas com cada pacote que é recebido em uma das suas interfaces.

Cada um dos *switches* SDN recebe as entradas de fluxo de um controlador através do que chamamos de *Southbound API*. Uma *Southbound API* é uma interface que interliga os planos de controle e de dados da rede SDN e possibilita um controlador receber e enviar mensagens dos *switches* SDN, além de realizar configurações neles.

Há diversas alternativas de *Southbound API's* [51] porém o *OpenFlow* [52, 4] é o protocolo mais largamente usado e adotado como padrão no SDN. Ele provê uma especificação comum para a implementação de *switches* SDN e também para um canal de comunicação entre o plano de controle e de dados [53].

O controlador, elemento fundamental na arquitetura SDN, de forma similar a um sistema operacional, abstrai os detalhes de baixo nível da interação com os dispositivos de encaminhamento. Adicionalmente, o controlador provê uma interface de programação dos *switches* SDN através da qual aplicações podem ser criadas para oferecer novas funcionalidades à rede.

Através da *Northbound API* as aplicações não precisam saber os detalhes sobre a topologia e se utilizam do controlador para programar o comportamento da rede. Conforme Braun [54], diferentemente da *Southbound API*, não há um padrão de interface entre as aplicações de rede e o controlador e, portanto, cada um implementa a sua *API*, sendo necessário conhecer os detalhes de uso de acordo com o controlador utilizado.

As aplicações SDN são as responsáveis por gerenciar as entradas de fluxo que são programadas nos dispositivos de rede, através do uso da API do controlador. Em geral, a responsabilidade de uma aplicação SDN é realizar uma função específica na rede. Dentre as diversas funções existentes podemos citar: *firewall*, balanceador de carga, *switches* de camada 2 e qualquer outra coisa que seja possível realizar através do uso da programação.

Através da programabilidade inserida pelo SDN, todo esse ecossistema torna possível as redes de computadores se adaptarem dinamicamente de acordo com as necessidades e o surgimento de novos conceitos.

3.2 O *switch OpenFlow*

Um *switch OpenFlow* é um software ou dispositivo de hardware que encaminha pacotes em um ambiente SDN. Ele é composto de uma ou mais tabelas de fluxos, que são utilizadas para o encaminhamento dos pacotes, além de um canal seguro com o controlador. O *switch* compõe o plano de dados e tem como responsabilidade realizar modificações nos pacotes conforme as entradas de fluxos que estão instaladas nas suas tabelas.

Dado que toda a inteligência está a cargo do plano de controle, o *switch* precisa

ser instruído pelo controlador através de mensagens que indiquem como deve ser o seu comportamento pois, caso contrário, nenhum pacote será entregue no destino. Além disso, o *switch* possui portas que são utilizadas para receber e entregar os pacotes na rede. A Figura 3.2 representa um *switch* com os componentes descritos acima.

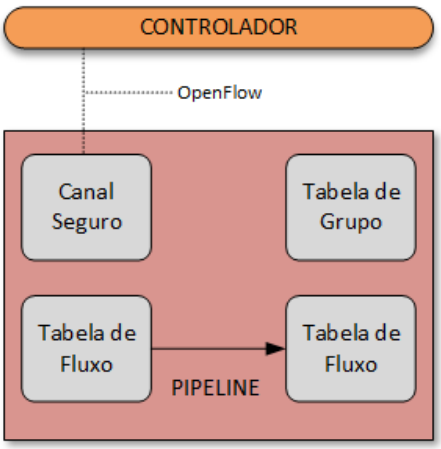


Figura 3.2: Componentes de um *switch OpenFlow* - Imagem adaptada de [4]

Os *switches* podem ser do tipo híbrido ou somente-*OpenFlow*. O *switch* híbrido suporta tanto a operação em um ambiente SDN quanto através da tradicional comutação de camada 2, mediante o seu plano de controle local, podendo decidir de acordo com o pacote recebido, qual dos dois processamentos será executado, enquanto que o *switch* somente-*OpenFlow* possui apenas as funcionalidade do plano de dados, sem a capacidade de tomar decisões de forma autônoma.

3.3 As portas *OpenFlow*

As portas *OpenFlow* são interfaces por onde os pacotes passam, bem como os componentes que realizam a ligação entre o processamento dos pacotes e o resto da rede. É através das portas que os pacotes entram no *switch* e, após o processamento, podem ser enviados para a rede novamente.

A quantidade de portas em um *switch* não é necessariamente igual ao número de interfaces que ele possui, pois algumas delas podem não dar suporte ao uso do *OpenFlow*, ao mesmo tempo que portas adicionais são definidas como ações a serem realizadas com os pacotes. Basicamente existem três tipos de portas: as físicas, as lógicas e as reservadas.

As portas físicas correspondem às interfaces de rede no hardware do *switch* e

portanto geralmente possuem um mapeamento um para um, já as portas lógicas são abstrações para as portas que não são físicas, como a porta de VLAN por exemplo, enquanto que as reservadas definem ações genéricas para o encaminhamento dos pacotes.

Embora muitas portas reservadas sejam definidas, este trabalho utiliza apenas algumas, que merecem atenção para que os aspectos da proposta sejam adequadamente compreendidos.

A porta *CONTROLLER* representa o canal de controle do *switch* com o controlador. Quando ela é utilizada como porta de saída, o *switch* cria uma mensagem do tipo *PacketIn* e envia o pacote ao controlador. Ao ser especificada como porta de entrada, a correspondência ocorre quando um pacote vindo do controlador é recebido.

Outra porta importante neste trabalho é a *LOCAL*, que possibilita a comunicação com o controlador por um canal separado do *OpenFlow* para troca de dados adicionais, como, por exemplo, pacotes BGP, enquanto que a porta *FLOOD* representa a operação de *flooding* (inundação), enviando o pacote por todas as portas, exceto a de entrada.

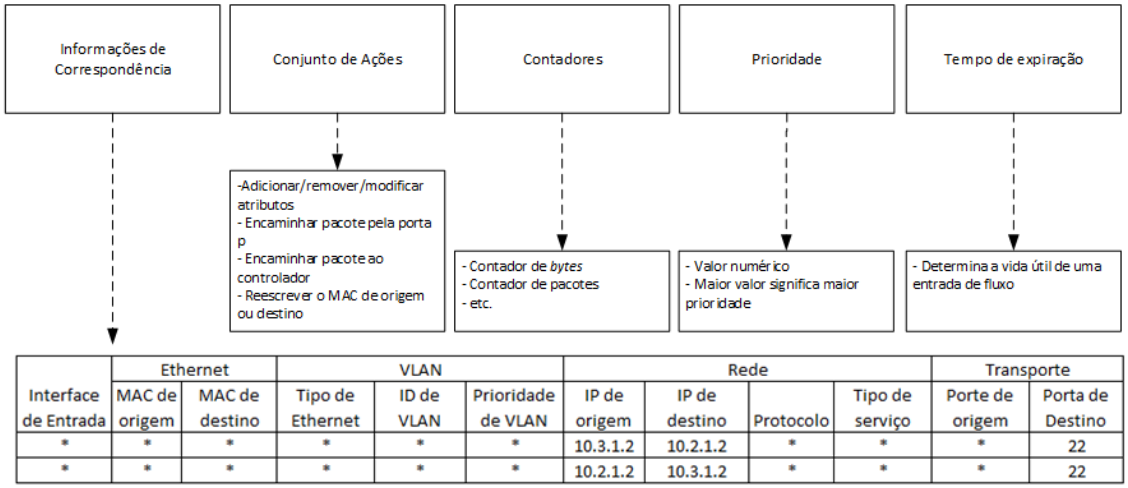
3.4 A tabela de fluxo

Um fluxo é uma sequência de pacotes que compartilham atributos e, portanto, possuem características em comum, que fazem um *switch* entende-los como iguais e trata-los da mesma maneira assim que eles são recebidos. Para separar esses fluxos de acordo com as suas especificidades, o *switch* os organiza em uma estrutura de dados em forma de tabela, chamada de tabela de fluxos, de modo que seja possível tomar diferentes ações conforme a correspondência com cada pacote recebido.

Essa tabela possui todos os fluxos que o *switch* conhece e sabe o que fazer no caso de recebimento de um respectivo pacote. Além disso, ela possui campos que não apenas determinam as informações de correspondência, mas definem outros aspectos como a prioridade de uma entrada em relação às outras, as ações que devem ser executadas com o pacote, alguns contadores e o seu tempo de expiração.

É possível a utilização de várias informações do cabeçalho de um pacote na definição de um fluxo como, por exemplo, IP de origem e de destino, MAC de origem e de destino, ID de VLAN ou até a porta lógica utilizada na camada de aplicação.

Isso torna a definição de fluxos extremamente flexível, possibilitando uma aplicação moldar o comportamento do plano de dados de uma maneira extremamente granular, de acordo com as suas necessidades. Um exemplo dessa flexibilidade na definição dos fluxos pode ser um conjunto de pacotes com os valores de IP de origem e destino 10.3.1.2 e 10.2.1.2 respectivamente, se comunicando através da porta 22. A Figura 3.3 mostra como poderia ser o estado da tabela de fluxos nessa situação.



	Ethernet		VLAN			Rede				Transporte	
Interface de Entrada	MAC de origem	MAC de destino	Tipo de Ethernet	ID de VLAN	Prioridade de VLAN	IP de origem	IP de destino	Protocolo	Tipo de serviço	Porte de origem	Porta de Destino
*	*	*	*	*	*	10.3.1.2	10.2.1.2	*	*	*	22
*	*	*	*	*	*	10.2.1.2	10.3.1.2	*	*	*	22

Figura 3.3: A tabela de fluxos do *OpenFlow*. Imagem adaptada de [5]

O estado da tabela de fluxos é gerenciado pelo controlador através da comunicação com o *switch* usando o canal seguro, de modo que fluxos são adicionados, removidos e atualizados conforme definido no plano de controle.

3.5 O processamento de pacotes no *OpenFlow*

Um *switch* possui tabelas de fluxo, sendo elas, numeradas sequencialmente a partir do valor 0 e compostas de diversas entradas de fluxo. O *switch* não precisa necessariamente possuir muitas tabelas porém necessita ter no mínimo uma, para que o processamento de pacotes seja realizado.

O processamento de pacotes no *OpenFlow* inicia analisando a primeira tabela de fluxos, a tabela 0, e caso alguma entrada de fluxo corresponda ao pacote recebido, a ação correspondente àquela entrada é realizada. Esta ação pode ser descartar o pacote, entrega-lo em alguma porta de saída ou até direcionar o pacote para uma outra tabela de fluxo, para a realização desse processo novamente.

Caso nenhuma das entradas de fluxo tenham correspondência com o pacote, o

comportamento a ser adotado é especificado através da entrada *table-miss*. Esta entrada define como deve ser processado um pacote que não tenha nenhuma correspondência com a tabela de fluxo na qual ela está. Neste caso, o pacote pode ser descartado, encaminhado para uma outra tabela, modificado e redirecionado através de alguma porta ou enviado ao controlador através do canal seguro. A Figura 3.4 ilustra o funcionamento desse mecanismo.

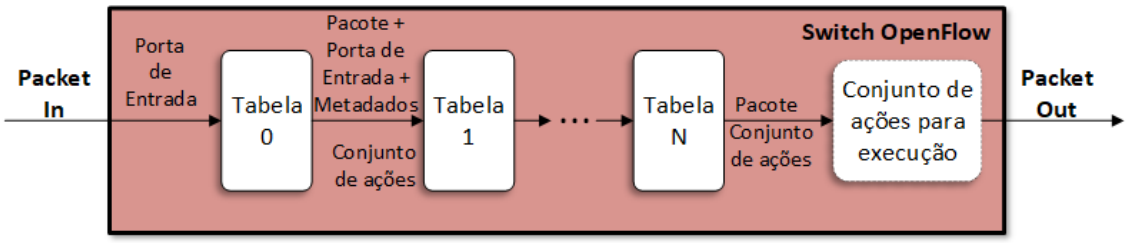


Figura 3.4: Processamento de pacotes via *OpenFlow* - Imagem adaptada de [4]

Para o processamento de pacotes, a especificação do *Openflow* [4] determina que a ação de encaminhamento para uma outra tabela de fluxo só pode ser realizada para tabelas com um número maior do que a atual onde o pacote está sendo processado. Por exemplo, caso um pacote esteja sendo processado na tabela de fluxos 1 e a entrada *table-miss* indique um redirecionamento para uma outra tabela, o destino só poderá ser a tabela 2 em diante. Sendo assim, a última tabela de um *switch* não poderá ter nenhuma ação de redirecionamento, uma vez que o processamento de pacotes não pode voltar na sequência das tabelas.

3.6 O canal seguro *OpenFlow*

O canal seguro é o caminho utilizado para a comunicação *OpenFlow* entre o *switch* e o controlador. Através dessa interface, o controlador pode gerenciar ou configurar o *switch*, receber eventos, enviar mensagens e obter dados estatísticos. Este canal geralmente é encriptado através de uma conexão utilizando TLS, porém, o seu uso não é obrigatório, logo, a comunicação pode ser executada apenas sobre o TCP sem nenhum tipo de criptografia [55].

As mensagens trocadas através do canal seguro podem ser de três tipos: controlador-para-*switch*, assíncronas ou simétricas. Para este trabalho, algumas delas serão sucintamente descritas a seguir em função da sua importância, uma vez que são utilizadas na implementação dos recursos necessários ao cumprimento dos objetivos da proposta.

As mensagens do tipo controlador-para-*switch* são utilizadas pelo controlador para gerenciar os aspectos do *switch*, além de executar modificações no seu estado. Dentre elas, temos as mensagens *FEATURES*, *ModifyState* e *PacketOut*. A primeira tem como função a obtenção da relação de recursos suportados por um *switch*, enquanto que a segunda possibilita a adição, modificação ou deleção das entradas na tabela de fluxo e a última é empregada para informar ao *switch* a lista de ações a serem aplicadas em um pacote.

Já as mensagens assíncronas são enviadas no sentido *switch*-controlador para informar sobre mudanças no seu estado, a chegada de um pacote ou até mesmo uma situação inesperada. Uma das mensagens assíncronas mais importantes é chamada *PacketIn*, onde o *switch*, ao receber um pacote que não possui correspondência com nenhuma entrada de fluxo, envia o seu cabeçalho ao controlador para receber instruções através de um *PacketOut*.

Embora não sejam formalmente explicitadas na proposta, as mensagens *HELLO* e *ECHO* são constantemente utilizadas pois, enquanto a primeira tem como função determinar a versão do *OpenFlow* utilizada assim que a conexão é estabelecida, a segunda permite a validação do funcionamento e a coleta de medições de latência ou largura de banda do canal seguro. Por serem enviadas por ambos os lados, elas são consideradas mensagens simétricas.

3.6.1 Comunicação entre o *switch* e o controlador

Após o estabelecimento do canal seguro, cada mensagem trocada entre o controlador e o *switch* é iniciada com o cabeçalho (tabela 3.1) *OpenFlow*. Este cabeçalho especifica a versão do protocolo que está sendo utilizado, o tipo da mensagem, o seu comprimento e o identificador da transação.

Versão	Tipo	Comprimento
Identificador da Transação (xid)		

Tabela 3.1: Cabeçalho das mensagens *OpenFlow*

Conforme Goransson [6], o diálogo entre um *switch* e o seu controlador pode ser compreendido nas fases de inicialização, operacional e monitoramento. A Figura 3.5 mostra, de uma forma simplificada, as mensagens que são trocadas nessas fases.

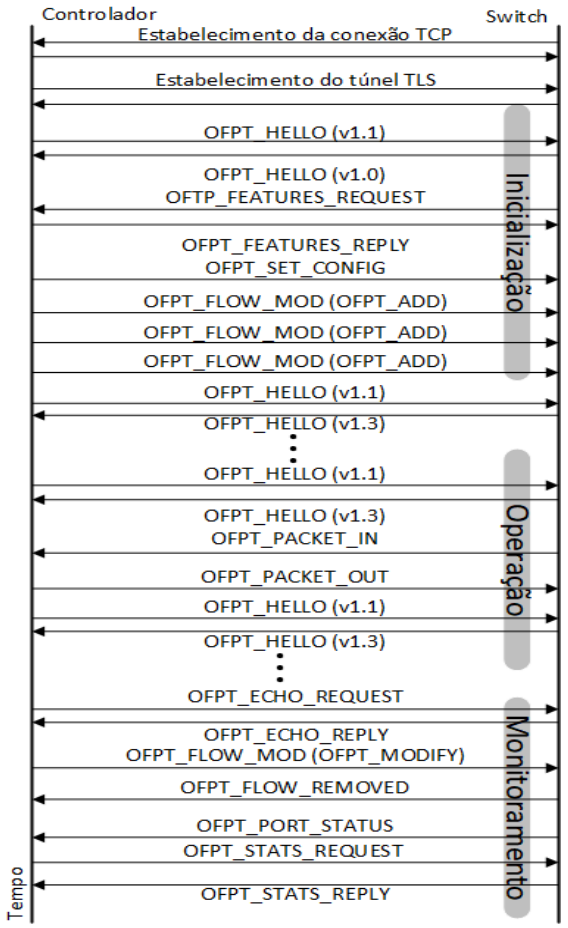


Figura 3.5: Comunicação entre o Controlador e o *Switch* no canal *OpenFlow*. Imagem adaptada de [6]

Na fase de inicialização, a conexão entre o *switch* e o controlador é iniciada e o canal seguro é estabelecido. O controlador verifica os recursos suportados e configura o *switch* através das mensagens *FEATURE_REQUEST* e *SET_CONFIG* além de adicionar entradas de fluxo com a operação *FLOW_MOD* (uma mensagem do tipo *Modify-State*)

Na fase de operação, ao receber pacotes que não correspondem a nenhuma entrada de fluxo em sua tabela, caso a entrada *table-miss* determine a entrega do pacote ao controlador, é gerada uma mensagem *Packet-In* para que o controlador defina o que fazer com o pacote através de uma mensagem *Packet-Out*, enquanto que, na fase de monitoramento, o *switch* se reporta ao controlador sobre mudanças de status e fluxos removidos. Apesar de terem sido separadas, as fases de operação e monitoramento se sobrepõe e coexistem de forma contínua.

3.7 O Controlador

O controlador é o elemento central em uma arquitetura SDN. Através dele, é possível ter uma visão global e definir tarefas de controle e gerenciamento da rede oferecendo assim, novas funcionalidades.

Conforme previamente mencionado na Seção 3.1, através de uma analogia, podemos comparar um sistema operacional com um controlador pois, de uma forma semelhante, este provê uma interface de programação (sistema operacional) para os *switches* (hardware do computador) [56]. Além disso, trabalhos [57, 58] envolvendo SDN já abordam os controladores como sistemas operacionais de rede.

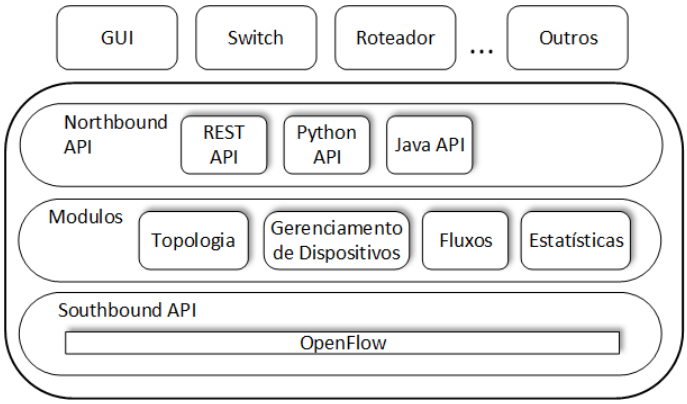


Figura 3.6: A anatomia de um controlador. Imagem adaptada de [6]

Conforme a Figura 3.6, além das *API´s Northbound* e *Southbound* um controlador possui outras funcionalidades, implementadas em forma de módulos entre as duas interfaces. Esses recursos são a descoberta de dispositivos de usuário, de dispositivos de rede, a manutenção das informações da topologia e o gerenciamento dos fluxos de dados.

Os controladores também podem oferecer uma *API REST* para disponibilizar acesso à serviços como, por exemplo, a obtenção de estatísticas e a modificação de configurações dos *switches* [59]. Em geral, as API´s disponibilizadas pelo controlador existem para fornecer uma abstração de alto nível para que as aplicações SDN possam realizar as operações sem necessariamente se preocupar sobre como funcionam os detalhes internos dos *switches*. Da mesma forma, o *OpenFlow* realiza essa abstração para fornecimento de serviços ao controlador.

Dada a importância de um controlador, há vários outros aspectos que devem ser levados em conta sobre eles. Tais particularidades são alvo de comparação por alguns trabalhos [60, 61, 62]:

- **Distribuídos x Centralizados:** Como o controlador é o principal elemento no SDN, ele precisa ter redundância para que, em caso de falha, a atividade de controle da rede não seja interrompida. Embora possuam a vantagem de fornecer tolerância a falhas no plano de controle, os controladores distribuídos possuem uma carga adicional para manter a consistência das informações entre os diversos controladores.

- **Suporte a plataformas:** Cada controlador pode oferecer a possibilidade de uso em uma ou mais plataformas e isso é fator de comparação em trabalhos [60] realizados. Em geral, os controladores podem ser utilizados na maioria dos sistemas Linux, porém há controladores que fornecem suporte à *Windows* ou MAC.

- ***Open Source* x Proprietário:** Há muitos controladores de código aberto, no entanto algumas empresas como *HPE*, *Cisco* e *Google* já desenvolveram os seus controladores e alguns deles não tiveram o seu código fonte liberado ou mesmo foram disponibilizados para uso.

- **Suporte a *Multithreading*:** Alguns controladores suportam múltiplas *threads*. Esse suporte permite a eles ter um ganho de performance e escalabilidade nas suas tarefas [63].

- **Suporte a versões do *OpenFlow*:** O suporte a versões do *OpenFlow* pode ser diferente entre os controladores. Essa fator é relevante com relação às funcionalidades suportadas pelo controlador, uma vez que, versões mais novas do protocolo suportam recursos não existentes nas anteriores.

- **Linguagem de programação:** Há controladores desenvolvidos nas mais diversas linguagens como *Python* [26], *Java* [64, 65, 66, 67], *Ruby* [68], *C++* [69] e *C* [70]. Além de impactar na facilidade de desenvolvimento, a linguagem de programação influencia diretamente a *Northbound API* que é disponibilizada e, até mesmo, a performance do controlador, conforme mostrado por Erickson [64].

3.8 As aplicações SDN

As aplicações SDN executam sobre o controlador e se comunicam com a rede através da *Northbound API*. Através da API, elas podem realizar muitas atividades

como, por exemplo, balancear o tráfego entre múltiplos caminhos, reagir a mudanças na rede ou até aplicar ações relacionadas à segurança.

Após o controlador iniciar, a aplicação usa a maior parte do tempo respondendo eventos vindos do controlador ou de fontes externas, conforme foi programada. Essas fontes externas podem ser outros sistemas, fontes de dados ou até mesmo pares BGP [6].

Dois tipos fundamentais de aplicações emergiram desde o início do SDN: as reativas e as proativas. Cada uma dessas formas possui as suas particularidades, pontos positivos e negativos e, de acordo com o objetivo da aplicação, uma ou outra forma de atuação pode ser mais apropriada.

3.8.1 Aplicações reativas x proativas

As aplicações reativas recebem periodicamente pacotes encaminhados pelo *switch* para que sejam processados e enviam instruções de volta descrevendo como eles devem ser manipulados. Caso seja necessário, além das instruções, entradas de fluxo podem ser inseridas ou modificadas.

Em geral, um *switch* gerenciado por uma aplicação reativa possui poucos fluxos em sua tabela de fluxos e os temporizadores dos fluxos inseridos são configurados para uma duração específica, de modo a limitar o crescimento da tabela.

Esse comportamento, pode gerar uma frequente ocorrência de recebimento de pacotes que não correspondam a nenhuma entrada de fluxo gerando diversas mensagens *Packet-In* por parte do *switch*. Uma possível resposta a isso é inserir uma entrada de fluxo para que, na próxima vez que os pacotes do mesmo fluxo forem recebidos, o *switch* saiba o que fazer com eles.

As aplicações proativas, em geral, se comunicam menos com o *switch*, que não precisa realizar tantas requisições ao controlador. Além disso, elas realizam configurações ou inserções de entradas de fluxos nos *switches*, de modo que estes já estejam preparados para entregar os pacotes antes que eles cheguem. Conforme descrito por Goransson [6], isso pode ser exemplificado pelo caso onde uma aplicação SDN recebe um evento de algum módulo de monitoramento externo e, diante disso, realiza ajustes nas configurações ou nas entradas de fluxo de um *switch*.

Nas Figuras 3.7 e 3.8, pode ser visto o funcionamento básico das aplicações reativas e proativas, respectivamente.

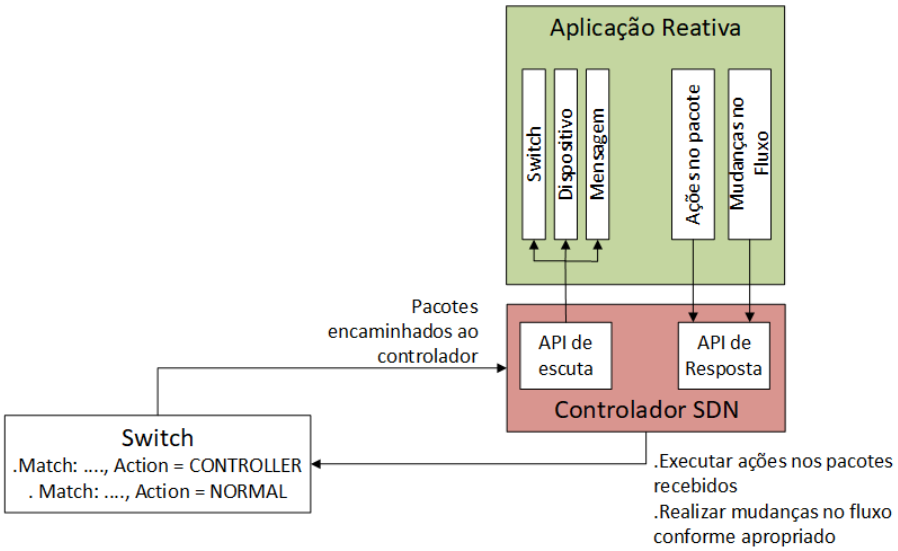


Figura 3.7: Aplicação SDN reativa. Imagem adaptada de [6]

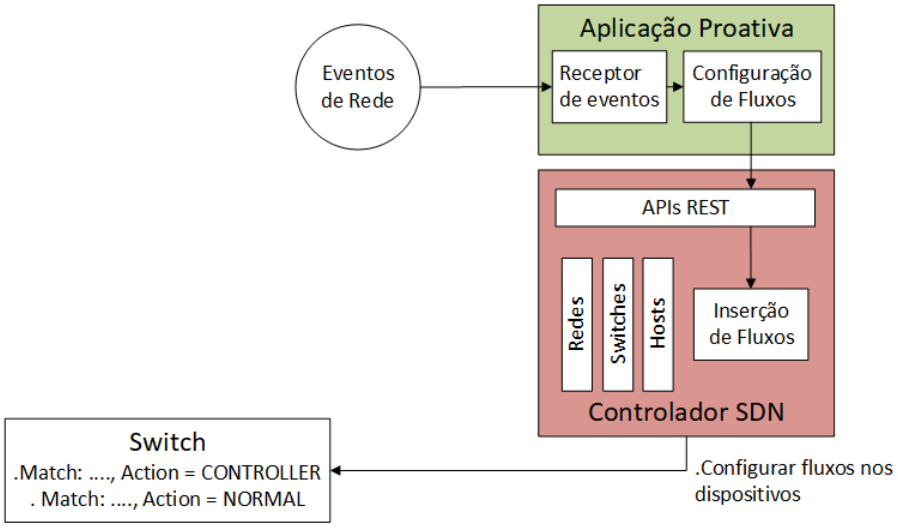


Figura 3.8: Aplicação SDN proativa. Imagem adaptada de [6]

As aplicações reativas costumam ser menos resilientes, no caso da perda de comunicação entre o *switch* e o controlador, devido a necessidade do *switch* saber o que fazer quando receber um pacote desconhecido. Uma outra vantagem do método proativo é que não há latência adicional na entrega dos pacotes quando os fluxos são populados. Em contra partida, aplicações proativas requerem um maior esforço de gerenciamento, enquanto possuem regras de fluxos mais gerais (*wildcard*) para atender a todas as rotas [71].

3.9 Síntese

Este capítulo fundamentou as redes definidas por *software*, um assunto relevante para o entendimento da proposta, discutindo brevemente como cada elemento participa na sua pilha de tecnologias.

4. A proposta

Este capítulo explica a proposta em uma abordagem *top-down*, iniciando através de uma descrição abrangente do funcionamento do *AS-Defender* para, em seguida, detalhar as suas fases de funcionamento e os seus mecanismos de detecção e mitigação do ataque. Ao final, é fornecida uma breve descrição sobre alguns trabalhos relacionados.

4.1 Visão geral

De acordo com o que foi mencionado no Capítulo 1,o objetivo deste trabalho é entender como uma arquitetura SDN, e os seus benefícios inerentes, podem auxiliar no combate ao ataque *prefix hijacking* direcionado aos sistemas autônomos. Para alcançar o objetivo proposto, este trabalho apresenta o *AS-Defender*, um protótipo[1] de aplicativo SDN para fazer parte do plano de controle de um AS de trânsito, cujo papel é identificar e mitigar o ataque *prefix hijacking* direcionado aos prefixos dos seus vizinhos.

Num contexto geral, o *AS-Defender* utiliza os dados do plano de controle BGP (mais especificamente as mensagens *UPDATE*) para detectar os sequestros de prefixos IP direcionados às redes dos seus vizinhos. Após isso, adota ações de mitigação na tentativa de atrair a maior quantidade de tráfego possível, com destino à faixa de IP sequestrada, para entrega-lo novamente ao vizinho atacado. O foco principal é normalizar a entrega do fluxo de dados para o AS afetado, evitando que os dados sejam atraídos pelo atacante e, consequentemente, o ataque seja realizado com sucesso. Além disso, o operador do *AS-Defender* pode configurar alguns aspectos dos mecanismos de defesa de modo que a proteção de cada vizinho possa ser feita

[1]Versão preliminar, reduzida, de um novo sistema de computador ou de um novo programa, para ser testada e aperfeiçoada

conforme as necessidades de cada um deles.

O protótipo é totalmente escrito em Python, uma linguagem de alta produtividade, reduzida curva de aprendizado e a mesma na qual o controlador *Ryu*, utilizado neste trabalho, foi escrito.

O funcionamento do aplicativo se dá em quatro fases: instanciação, operação, monitoramento e mitigação. Na fase de instanciação são realizadas algumas configurações iniciais no *switch*, possibilitando a comunicação do aplicativo com os ASes vizinhos. Uma vez que toda a comunicação inicial esteja estabelecida, inicia-se a fase de operação onde o aplicativo recebe as requisições do *switch* e determina as ações a serem tomadas com base nas informações do plano de controle. Paralelamente a operação, o monitoramento das mensagens *UPDATE*, do plano de controle BGP, é realizado para verificar a existência de um ataques contra os prefixos monitorados. Caso um ataque seja detectado, a fase de mitigação é acionada para que contra medidas sejam adotadas, na tentativa de redirecionar o fluxo alvo para o caminho correto.

Este trabalho utiliza alguns termos que precisam ser clarificados para o bom entendimento do funcionamento da solução proposta nas próximas seções. O AS atacante é o sistema autônomo que está realizando o ataque *prefix hijacking* dentro da topologia, da mesma forma que o AS vítima é um sistema autônomo, protegido pelo *AS-Defender*, que está tendo o prefixo sob sua propriedade anunciado pelo atacante. Os *hosts* são maquinas que estão posicionadas internamente em alguns dos ASes com algumas finalidades de teste. O *switch* é o *datapath* controlado pelo *Ryu*, que é comandado pelo *AS-Defender*. O coletor de rotas é um roteador BGP inserido em um dos ASes através de uma conexão iBGP para, ao receber uma mensagem *UPDATE*, disponibiliza-la através de uma interface *WebSocket*. Clientes são os ASes vizinhos que fazem sessão BGP com o *AS-Defender* e podem ser protegidos por ele. A Figura 4.1 mostra a definição de cada um dos componentes descritos anteriormente na topologia BGP.

O *AS-Defender*, como uma aplicação SDN, localiza-se acima do controlador consumindo a sua *Northbound API*, através da qual, é possível realizar as configurações no *switch*. Entre o *switch* e o controlador além do canal *OpenFlow*, há um canal de comunicação para que as mensagens dos vizinhos sejam entregues ao plano de controle e as sessões eBGP possam ser estabelecidas.

É através da interface *WebSocket* que o mecanismo de monitoramento analisa as

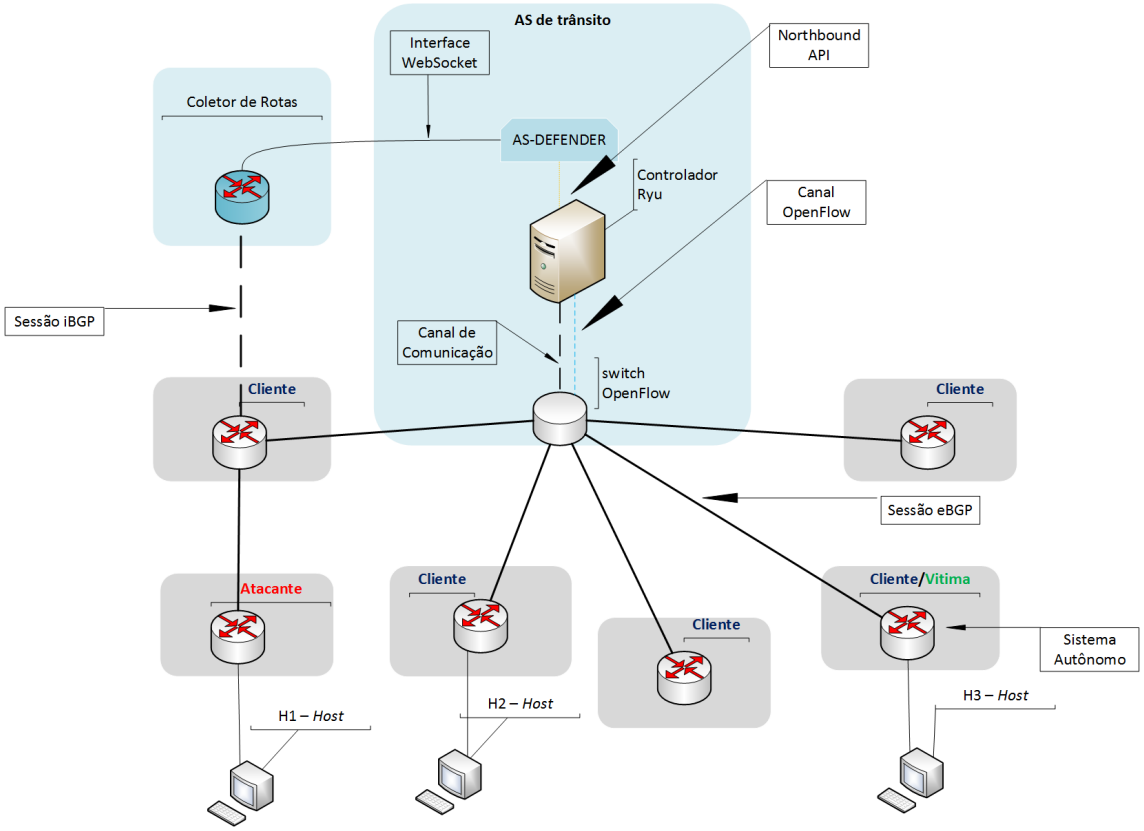


Figura 4.1: Topologia BGP com os componentes da solução

mensagens de anúncios BGP para verificar se o prefixo de algum dos seus clientes foi atacado. As ações de mitigação são realizadas no plano de controle onde ações são requisitadas ao controlador através da *Northbound API*.

4.2 O coletor de rotas

O coletor de rotas, ilustrado na Figura 4.1, é o principal componente envolvido na coleta de anúncios BGP. Ele é o responsável por, ao receber as mensagens *UPDATE* através da conexão iBGP, entrega-las ao mecanismo de monitoramento do *AS-Defender* em praticamente tempo real.

Esse mecanismo é possível através da implementação de um servidor *WebSocket* que, ao coletar os anúncios recebidos, realiza a entrega das mensagens através de uma interface com a *thread* de monitoramento, que implementa um cliente *WebSocket*.

O protocolo *WebSocket* [72] permite a comunicação bidirecional entre cliente e servidor através de um *socket* TCP. Através dele, após o estabelecimento de uma conexão entre o cliente e o servidor, ambos os lados podem enviar mensagens em

um canal *full-duplex* tornando possível a análise de dados com bastante rapidez.

O coletor de rotas executa um processo BGP que apenas existe para receber anúncios e disponibiliza-los aos seus clientes. Assim que um anúncio é recebido, seus dados são jogados na saída padrão que é constantemente monitorada pelo servidor *WebSocket*. Ao verificar uma nova mensagem, o servidor coleta os dados, formata em uma estrutura JSON[2] e entrega para a *thread* de monitoramento. O papel do coletor de rotas na topologia é representar um mecanismo semelhante ao comportamento dos serviços de disponibilização de dados de roteamento BGP existentes na internet, como o *Route Views* [21] e o RIPE [22].

4.3 A *thread* de monitoramento

A *thread* de monitoramento é criada pelo processo do *AS-Defender* para realizar o monitoramento dos anúncios BGP paralelamente às outras atividades. Ao realizar uma conexão com o coletor de rotas são passados os dados do canal onde se deseja conectar e também o prefixo para monitoramento. Não foi possível passar explicitamente o prefixo alvo de monitoramento uma vez que o coletor de rotas permitia passar apenas um prefixo por conexão e assim poderia ser necessária a criação de diversas conexões para o monitoramento de muitos prefixos. Consequentemente, a verificação se o anúncio é relacionado a um prefixo monitorado ficou a cargo do algoritmo de detecção do ataque realizado pelo *AS-Defender*.

Como uma responsabilidade adicional, ao receber um anúncio, a *thread* verifica se este é legítimo ou não e, caso seja um anúncio falso, avisa ao processo principal para que as ações de mitigação sejam realizadas. Para isso, ela possui acesso à uma área compartilhada com o processo do *AS-Defender*, que é usada para a transmissão de mensagens assim que um ataque é identificado. Essas mensagens contém dados que são necessários para a realização da mitigação do ataque além de configurações do cliente afetado.

4.4 As fases de funcionamento

Conforme descrito na Seção 4.1, o funcionamento do *AS-Defender* pode ser compreendido em quatro fases. Os nomes das fases fazem uma alusão em relação a

[2]É um formato de troca de mensagens entre sistemas independente de linguagem. Costuma ser bastante utilizado também como uma forma de armazenamento de configurações de sistemas.

comunicação entre um *switch* e um controlador definidas na Seção 3.6.1 porém são relacionadas ao comportamento da aplicação SDN e ao seu mecanismo de detecção e mitigação do *prefix hijacking*.

A separação entre as fases é em grande parte conceitual uma vez que em diversos momentos o comportamento do *AS-Defender* tem relação com mais de uma fase simultaneamente. As fases de funcionamento foram chamadas de instanciação, operação, monitoramento e mitigação.

4.4.1 Instanciação

Na primeira fase, chamada de instanciação, são realizadas algumas configurações iniciais. Inicialmente são populadas algumas estruturas de dados relacionadas às rotas para os ASes vizinhos e aos dados do *switch*. São carregados, no plano de controle, os dados das interfaces do *switch*, sua tabela ARP e informações necessárias para a formação da sua tabela de encaminhamento. Os dados das interfaces do *switch* são obtidos através de um arquivo do tipo YAML, que é carregado na inicialização do aplicativo. Além disso, as informações dos clientes a serem protegidos são carregadas à partir de um arquivo JSON configurado previamente.

Adicionalmente, são instaladas algumas entradas de fluxo proativamente no *switch*. Tais entradas são necessárias para possibilitar a comunicação entre o plano de controle, onde está a inteligência do *AS-Defender*, e os demais vizinhos. Essas entradas de fluxo, caso não fossem instaladas, impossibilitariam o estabelecimento das sessões BGP entre o *AS-Defender* e os ASes vizinhos bem como a troca de pacotes ARP e IP na topologia. A tabela 4.1 mostra as regras instaladas proativamente na fase de instanciação.

Tabela de Fluxo	**Prioridade**	**Correspondência**	**Ação**
0	1	arp,in_port=1	LOCAL
0	1	arp,in_port=2	LOCAL
0	1	arp,in_port=3	LOCAL
0	1	arp,in_port=4	LOCAL
0	1	arp,in_port=5	LOCAL
0	1	ip,nw_dst=ip_1	LOCAL
0	1	ip,nw_dst=ip_2	LOCAL
0	1	ip,nw_dst=ip_3	LOCAL
0	1	ip,nw_dst=ip_4	LOCAL
0	1	ip,nw_dst=ip_5	LOCAL
0	0	*	CONTROLLER:65535

Tabela 4.1: Entradas de fluxo instaladas no *switch* na fase de instanciação

As cinco primeiras regras são para habilitar a entrega de pacotes ARP, que entrarem pelas interfaces do *switch*, para o controlador. A ação de entrega LOCAL significa que os pacotes ARP serão enviados diretamente ao controlador, através de um canal separado, para tratamento pelo processo BGP que está sendo executado.

As regras seguintes são referentes a entrega dos pacotes IP com destino a uma das interfaces do *switch*. Da mesma forma que as anteriores, estas são entregues ao controlador através da interface LOCAL para o devido tratamento. No campo de ações, as informações ip_1 até 5 devem ser entendidas como o IP de cada uma das interfaces do *switch*. A última entrada é a *table-miss* que define a entrega ao controlador. Todas as regras instaladas na fase de instanciação não possuem tempo de expiração e , portanto, perduram por todo o funcionamento do *AS-Defender* para sempre garantir a comunicação entre o plano de controle a os demais ASes.

Todas essas regras iniciais são inseridas após a realização do *handshake* do *OpenFlow* e da verificação de recursos do *switch* através da mensagem *FEATURES*. Em seguida, o processo BGP é iniciado e passa a buscar conexão com seus vizinhos para o estabelecimento das sessões eBGP e o inicio das operações no roteamento inter-domínios. A Figura 4.2 ilustra o funcionamento da fase de instanciação.

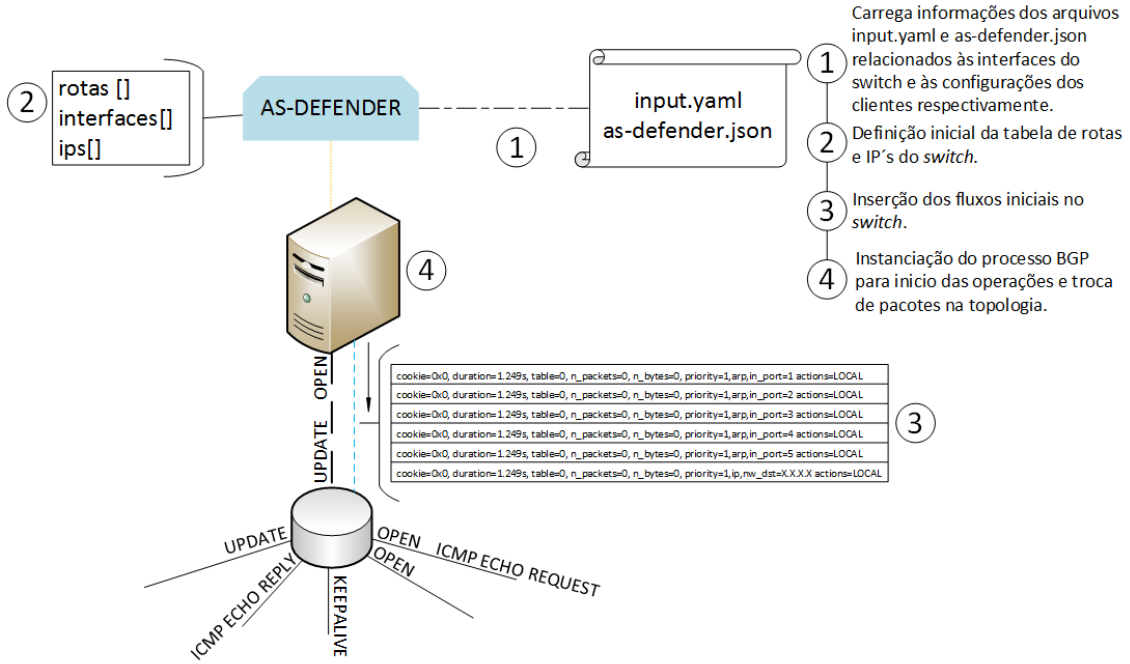


Figura 4.2: Instanciação do *AS-Defender*

A etapas da fase de instanciação foram numeradas de 1 a 4 com o intuito de definir uma cronologia e sequenciamento de ações na qual o *AS-Defender* executa

as operações.

4.4.2 Operação

A etapa de operação é descrita pelo recebimento de várias mensagens (ICMP, BGP, ARP etc..) pelo *switch* e a realização da ação correspondente ao pacote recebido. Caso o *switch* não saiba o que fazer com o pacote, realizará o envio de uma mensagem *PacketIn* ao controlador. O *AS-Defender* captura o evento do *PacketIn* enviado e, de acordo com as informações armazenadas no plano de controle, irá inserir uma entrada de fluxo no *switch* para que os próximos pacotes referentes àquele fluxo não necessitem ser enviados ao controlador. Diferentemente dos fluxos iniciais, estes possuem um tempo de expiração que quando é alcançado a entrada é retirada do *switch*. Os fluxos na fase de operação não poderiam ser inseridos sem tempo de expiração por que forçariam a existência de rotas estáticas para os prefixos na topologia, o que iria de encontro a dinamicidade do funcionamento do BGP.

Todo esse funcionamento habilita a comunicação na topologia e, a etapa de operação, permeia todo o ciclo de funcionamento do *AS-Defender* para que os ASes possam enviar trafego uns aos outros. A fase de operação serve como um suporte a qualquer outra atividade desempenhada pelo *AS-Defender* pois, sem a devida conectividade entre os ASes, não seria possível realizar as demais atividades. A Figura 4.3 demonstra a operação do *AS-Defender*.

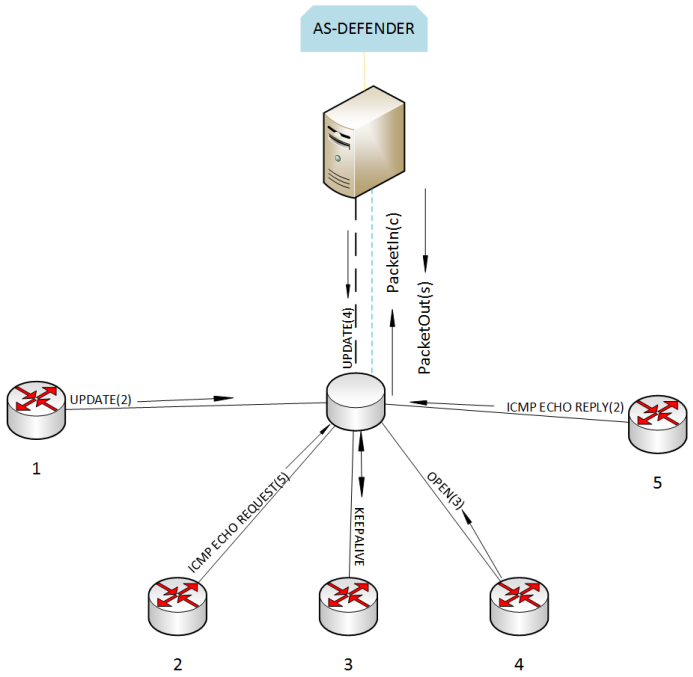


Figura 4.3: A fase de operação

As setas indicam os sentidos das mensagens e os números ou letras entre parênteses determinam o seu destino onde *c* e *s* significam controlador e *switch* respectivamente. Todo o comportamento de inserção dos fluxos durante a fase de operação é realizado de forma reativa, ou seja, somente na presença de uma mensagem *PacketIn*.

Os fluxos inseridos na fase de operação foram definidos com um tempo de expiração de 10 segundos. Esse tempo não teve nenhum embasamento teórico, mas foram obtidos empiricamente, de modo que o *AS-Defender* não demorasse demais para expirar os fluxos a ponto atrapalhar o mecanismo de mitigação em algumas situações, ao mesmo tempo que não gerasse muitas mensagens *packetIn*.

4.4.3 Monitoramento

A fase de monitoramento inicia desde o momento em que o aplicativo finaliza a sua instanciação e a *thread* de monitoramento é iniciada. O coletor de rotas e a *thread* de monitoramento formam o mecanismo responsável por receber os anúncios da topologia, formata-los, verificar os dados e detectar o ataque.

A *thread* de monitoramento se conecta ao coletor de rotas na porta 5000, fornece os dados de inscrição para a conexão via *WebSocket* e passa para um estado de espera onde aguarda a entrega dos dados pelo servidor. Assim que os anúncios são recebidos através da topologia, o coletor de rotas realiza uma formatação neles e os entrega à *thread*, que realiza a verificação do ataque.

A Figura 4.4 ilustra a sequência de funcionamento desde o recebimento de um anúncio pelo AS até a entrega da mensagem para a *thread* que analisará a ocorrência do ataque.

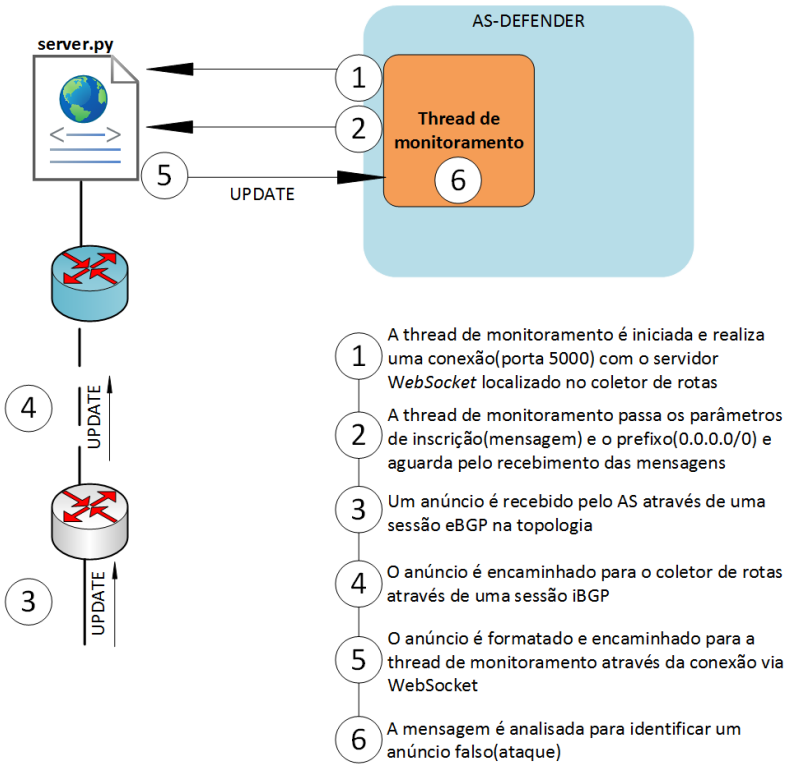


Figura 4.4: O Mecanismo de monitoramento do *AS-Defender*

A comunicação entre o coletor de rotas e a *thread* é feita através de um canal separado da topologia. Esse canal de comunicação é necessário para evitar qualquer influência de um ataque na topologia sobre o mecanismo de monitoramento.

4.4.4 Mitigação

A fase de mitigação inicia-se quando a *thread* de monitoramento identifica um ataque realizado ao prefixo de propriedade de um dos clientes. Imediatamente, são coletados alguns parâmetros e o mecanismo de mitigação do ataque é acionado, de acordo com as configurações do cliente atacado.

Conforme mencionado na Seção 4.3, para que a *thread* de monitoramento e o processo do *AS-Defender* possam se comunicar, foi criada uma área compartilhada onde, no momento da detecção do ataque, dados são inseridos para que o processo principal possa coleta-los e proceder com o processo de mitigação.

Esse mecanismo se fez necessário devido a questões tecnológicas envolvendo o Ryu. Assim que uma aplicação SDN baseada no Ryu é executada, o próprio *framework* executa o código criado sem explicitamente instanciar nenhum objeto, então, para utilizar a API do controlador sem precisar saber o nome do objeto instanciado

foi necessário realizar uma passagem de dados para uma área intermediária onde o processo principal consulta constantemente. Ao encontrar dados nessa área compartilhada, o *AS-Defender* entende que houve um ataque e coleta os dados existentes, que orientam sobre a mitigação a ser realizada.

A transferência de dados entre a *thread* e o processo principal é unidirecional, de modo que, apenas a *thread* transfere dados ao processo, não sendo realizado o processo inverso. Isso foi basicamente uma decisão de projeto uma vez que esse comportamento é suficiente para garantir o processo de mitigação e tal operação não demanda envio de confirmação para a *thread*, que continuará monitorando os anúncios. A Figura 4.5 ilustra o sequenciamento de ações na fase de mitigação.

Inicialmente a *thread* de monitoramento identifica o *prefix hijacking* e transfere os dados referentes ao ataque para a área compartilhada. Em seguida, o processo principal constata a existência de novos dados e os coleta para realizar a mitigação. O tipo de mitigação dependerá dos dados passados pela *thread* de monitoramento à partir das configurações do arquivo *as-defender.json*, carregadas na fase de instanciação.

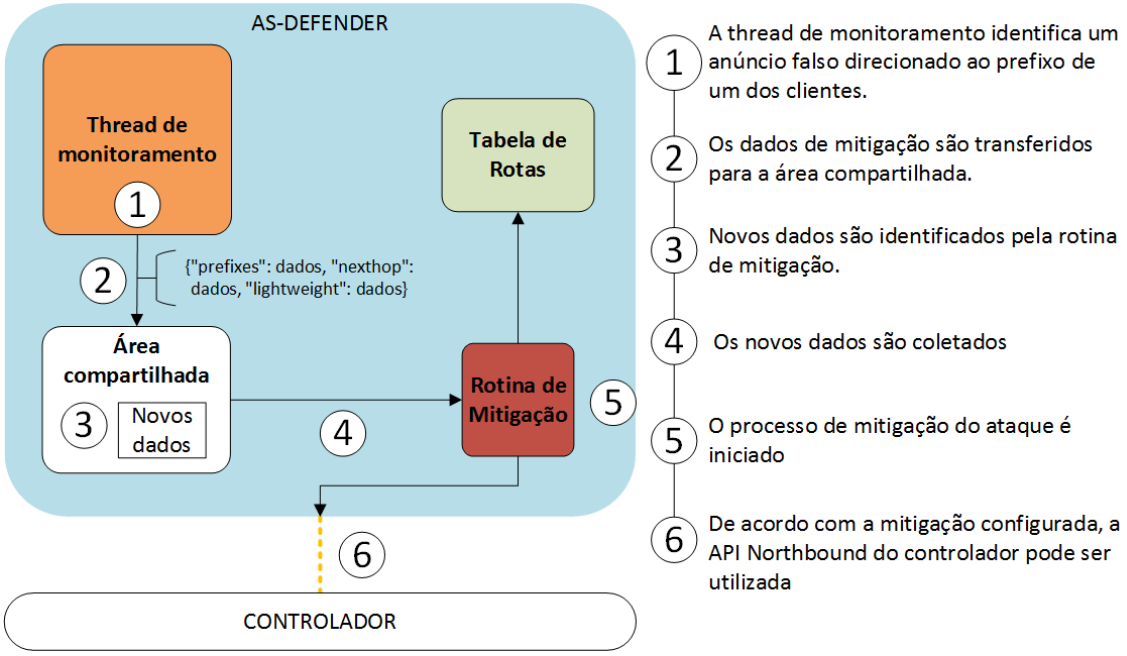


Figura 4.5: A fase de mitigação

O objetivo principal da fase de mitigação é, na presença de um ataque direcionado ao prefixo de pelo menos um dos clientes, fornecer novamente um caminho válido para o prefixo atacado ao maior número de ASes possível.

Campo	Descrição
ASN	É o número do sistema autônomo protegido
PREFIX	Define o prefixo de rede que deve ser protegido
NEIGHBORS	É uma lista com o ASN de todos os vizinhos do cliente
SUBPREFIXES	É uma lista com os sub-prefixos que serão utilizados pela rotina de mitigação do ataque
MOAS	Contém o ASN do *AS-Defender*.
MOASNEIGHBORS	Contém a lista de ASN´s dos vizinhos do *AS-Defender*
NEXTHOP	Contém o IP que o cliente utiliza para realizar a sessão eBGP com o *AS-Defender*
ANNOUNCE	É uma *flag* utilizada para o mecanismo de mitigação.

Tabela 4.2: Campos do arquivo *as-defender.json*

4.5 A metodologia de detecção do ataque *prefix hijacking*

Para detectar o ataque *prefix hijacking*, foi desenvolvido um algoritmo baseado no método de detecção descrito por SERMPEZIS [10]. A detecção do ataque é baseada no monitoramento dos anúncios BGP que ocorrem na topologia recebidos pelo coletor de rotas. O principal componente do anúncio utilizado na detecção do ataque é o atributo *AS PATH*, descrito na Seção 2.1. Através da análise do *AS PATH* é verificado se um prefixo foi sequestrado, o tipo de ataque de prefixo e o AS atacante. Além disso, com o uso de informações adicionais do cliente e da mensagem de anúncio, podemos verificar o prefixo atacado e o cliente afetado pelo ataque.

Este método de detecção não possui falsos positivos uma vez que se baseia no uso de dados do cliente e efetua comparações entre essas informações e o anúncio recebido para identificar o ataque.

A identificação é realizada com o auxilio do arquivo *as-defender.json* que contém diversas informações sobre os clientes que devem ser protegidos. Essas informações podem ser ajustadas para que outros clientes sejam protegidos ou para atender a mudanças nas configurações dos clientes já existentes. Os itens da Tabela 4.2 são inseridos em uma lista de dicionários e valem para cada cliente.

Para identificar o ataque, ao receber um anúncio, o algoritmo percorre as informações de cada cliente para saber se algum deles teve o seu prefixo atacado e realizar as ações de mitigação específicas configuradas em cada um deles.

Após ser formatado pelo servidor *WebSocket*, a *thread* de monitoramento recebe a mensagem do anúncio estruturada conforme a Tabela 4.3.

Campo	Descrição
TIMESTAMP	Horário da mensagem do anúncio em formato POSIX
HOST	*Host* que enviou a mensagem. Aqui sempre aparece o servidor *WebSocket*
PREFIX	Informa o prefixo anunciado
PEER	Informa o IP do vizinho que enviou a mensagem ao servidor *WebSocket*
PATH	Contém o campo *AS PATH* do anúncio
TYPE	Contém o tipo da mensagem.

Tabela 4.3: Estrutura de um anúncio recebido pela *thread* de monitoramento

Caso um AS anuncie mais de um prefixo em um mesmo anúncio, o servidor *WebSocket*, através da formatação, separa cada um dos prefixos anunciados em uma mensagem e as envia à *thread* de monitoramento. Isso significa que o campo *PREFIX* sempre conterá apenas um prefixo para cada mensagem recebida.

O método de detecção visa analisar o primeiro e o segundo valor do campo *AS PATH* para identificar se, em alguma das duas posições, há um ASN que configure um anúncio falso direcionado ao prefixos de um dos clientes.

Uma vez que o valor mais à direita representa o AS que anunciou primeiramente um determinado prefixo, este valor só pode ser preenchido com o ASN do proprietário do prefixo. Seguindo a mesma lógica, o segundo valor deve conter o ASN de um dos vizinhos do proprietário. Se alguma dessas condições não forem satisfeitas, para um anúncio relacionado ao prefixo de um dos clientes, o mecanismo de detecção identificará o *prefix hijacking*.

Para realizar essa tarefa, o prefixo de cada um dos clientes é armazenado em uma lista e, para cada mensagem recebida pela *thread* de monitoramento, toda a lista de prefixos é percorrida para saber se o anúncio está relacionado a algum dos clientes. Assim que o mecanismo de detecção identifica que o anúncio se refere a um dos prefixos listados o algoritmo de detecção é executado para saber se está ocorrendo um ataque.

De acordo com o mecanismo de mitigação adotado por cada cliente, pode ser necessário que o *AS-Defender* precise anunciar o prefixo do cliente em seu nome e, para isso, o algoritmo de detecção verifica que o anúncio está sendo realizado pelo próprio mecanismo de mitigação, não emitindo um alerta de ataque. Essa verificação

ocorre da mesma forma que no caso do proprietário do prefixo. Caso tal processo não fosse realizado, o mecanismo de detecção identificaria uma ação da própria rotina de mitigação como se fosse um ataque e isso geraria uma sequência de *loops* entre detecção e mitigação indefinidamente.

Os campos *TIMESTAMP*, *HOST*, *PEER* e *TYPE* não são utilizados para o mecanismo de detecção porém são enviados pelo servidor de acordo com o processo de formatação e podem ter utilidades futuras conforme o mecanismo de detecção receber novos recursos. Da mesma forma os campos *NEXTHOP* e *ANNOUNCE* não são consultados para a identificação do ataque apenas para a sua mitigação.

O algoritmo de detecção possui uma sequência de atuação, sendo assim, caso mais de um tipo de ataque seja identificado no mesmo anúncio, o primeiro a ser encontrado irá gerar a mensagem de detecção e o gatilho para as ações de mitigação. A Figura 4.6 mostra um fluxograma que representa o funcionamento do algoritmo de detecção do *prefix hijacking*.

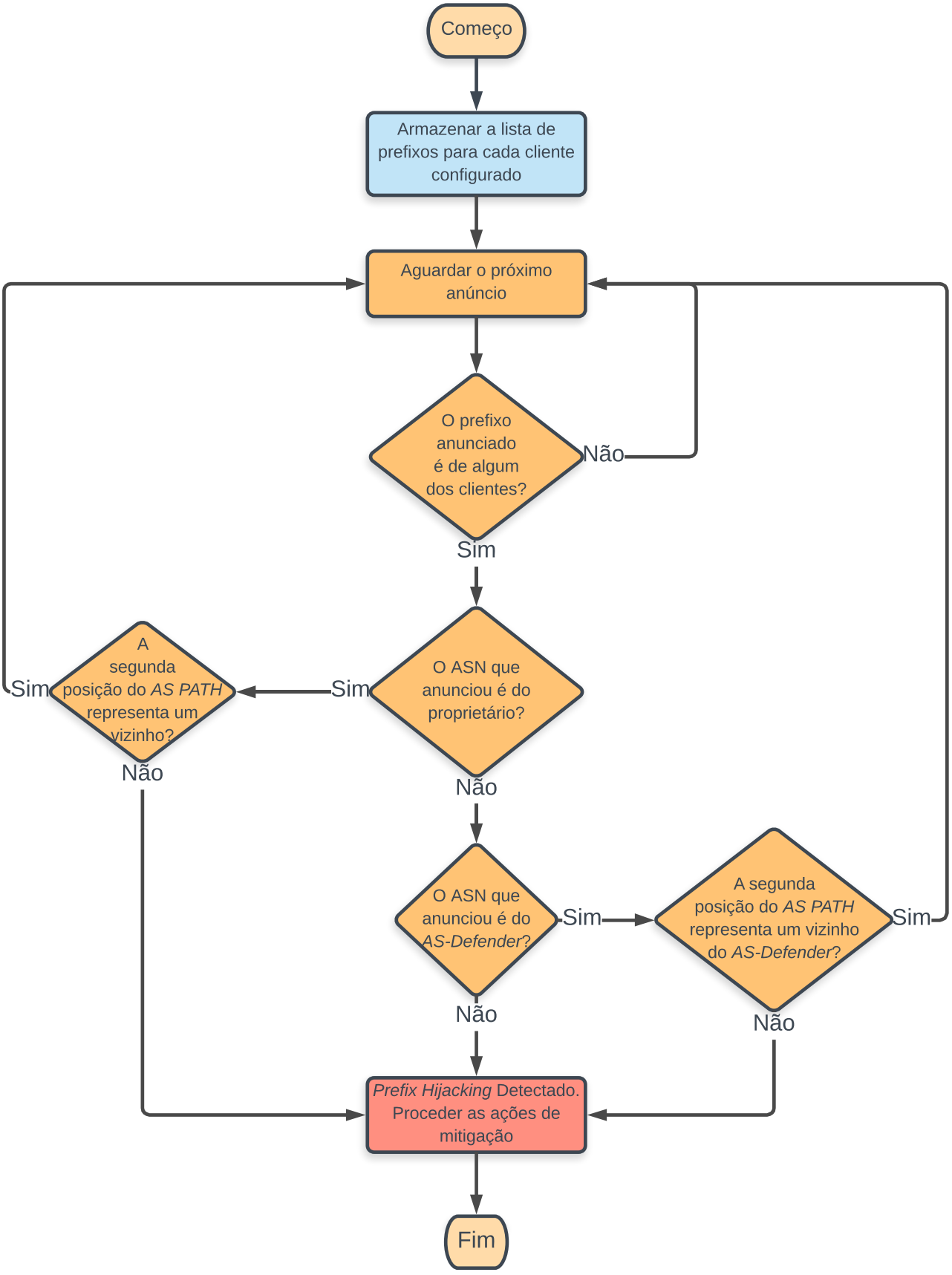


Figura 4.6: Fluxograma de Detecção do Ataque

4.6 O processo de mitigação

O processo de mitigação inicia após a identificação do ataque pela *thread* de monitoramento. A mitigação tem como objetivo reduzir, ao máximo possível, os danos causados pelo ataque. Deve ser entendido por redução de dano do ataque, o ato de proporcionar ao máximo de ASes possível o acesso ao prefixo do AS vítima mesmo após este ter sido atacado.

De uma forma geral o mecanismo de mitigação busca redirecionar o tráfego alvo para o AS vítima com o objetivo de manter a comunicação e o acesso aos seus serviços disponível. Desde que o aplicativo esteja adequadamente configurado com as informações dos clientes, o mecanismo de mitigação irá atuar de forma automática assim que um ataque for identificado.

Após a detecção do ataque, a *thread* de monitoramento encaminha os dados dos campos *SUBPREFIXES*, *NEXTHOP* e *ANNOUNCE*, descritos na Tabela 4.2, para o mecanismo de mitigação através da área compartilhada.

Para que o *AS-Defender* saiba qual deve ser o tipo de mitigação adotada, o campo *ANNOUNCE* é verificado. O campo *ANNOUNCE* só pode assumir os valores "0" ou "1" que significam desativado e ativado respectivamente. Se a *flag ANNOUNCE* estiver desativada, a mitigação sem anúncio será realizada, caso contrário, como uma tarefa adicional, os subprefixos da vítima, obtidos através do valor *SUBPREFIXES*, serão anunciados.

4.6.1 A mitigação sem anúncio

A mitigação sem anúncio possui o objetivo de impedir ou reduzir as proporções do *prefix hijacking* sem influenciar nas informações de roteamento BGP recebidas pelos demais ASes. Nesse sentido, apenas as informações de roteamento internas ao *AS-Defender* são ajustadas após a ocorrência do ataque, fornecendo um leve ajuste e o redirecionamento do tráfego. Diante disso, apenas o tráfego que efetivamente for roteador através do *AS-Defender* será redirecionado para a vítima uma vez que não é adotada nenhuma ação externa. Sob a visão dos demais ASes, nenhum anúncio, conexão ou comando é recebido e eles continuam realizando o roteamento dos dados utilizando as rotas existentes em sua tabela de encaminhamento, incluindo a rota recebida a partir do ataque. A Figura 4.7 mostra o funcionamento da mitigação sem anúncio.

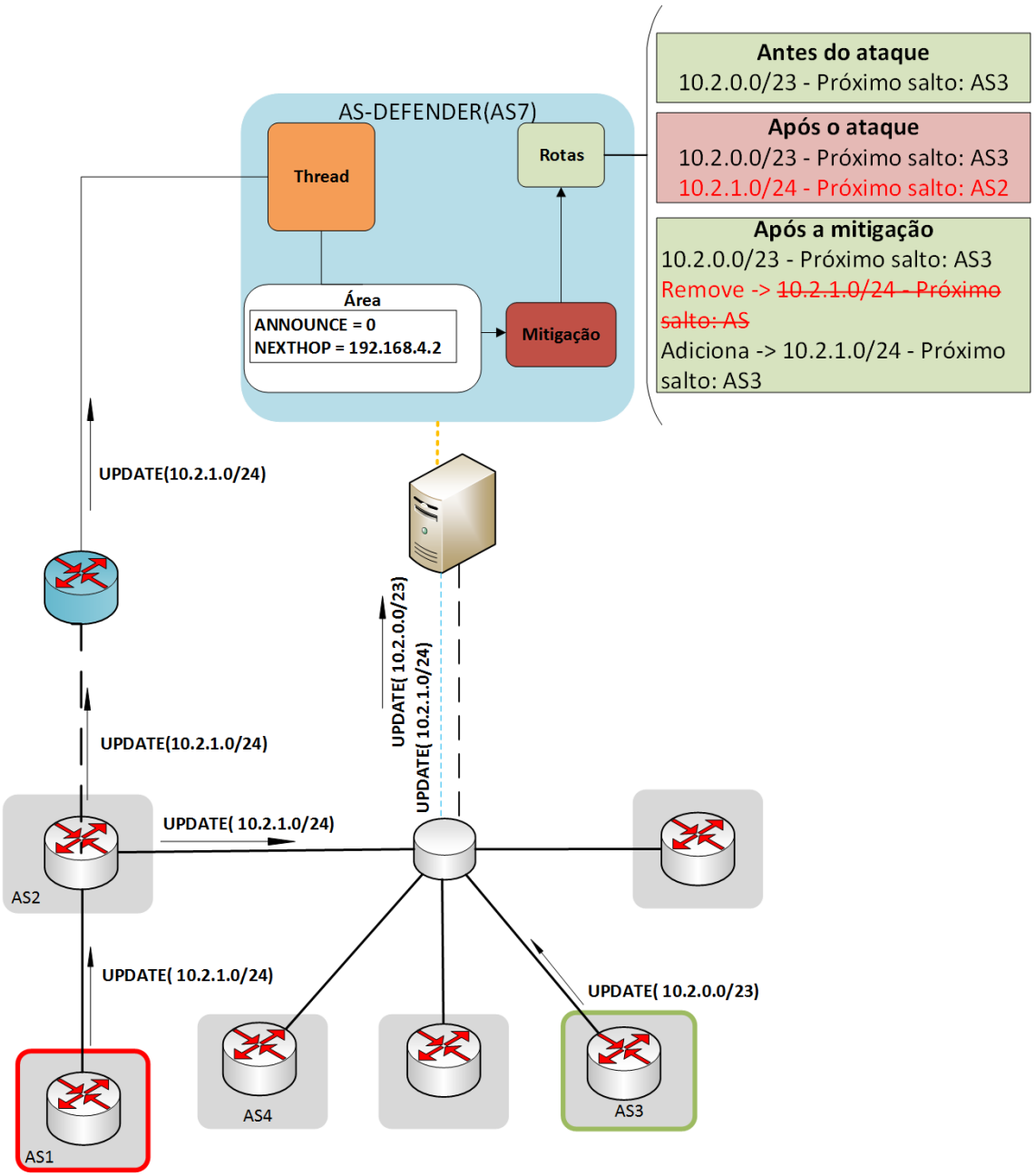


Figura 4.7: Funcionamento da mitigação sem anúncio

No exemplo ilustrado, o *AS3* (cliente/vítima) anuncia o prefixo 10.2.0.0/23, de sua propriedade, enquanto o *AS1* (atacante) realiza um ataque em um dos sub-prefixos (10.2.1.0/24) de propriedade da vítima. Quando o anuncio falso chega ao *AS-Defender*, este atualiza a sua tabela de rotas com o anuncio falso, porém, assim que o ataque é identificado pela *thread* de monitoramento, a rotina de mitigação verifica os dados recebidos e aplica uma correção na sua tabela, retirando a rota falsa e adicionando uma nova rota direcionando o prefixo atacado para o cliente. A inserção da nova rota direcionada para o cliente é possível através do valor do parâmetro *NEXTHOP*, encaminhado ao mecanismo de mitigação.

Após o ajuste, o mecanismo não insere nenhum regra de fluxo no *switch*, ao invés disso, o tempo de expiração da entrada existente é aguardado, e a partir do recebimento do próximo *PacketIn*, a entrada de fluxo correta é inserida. Esse comportamento gera um inconsistência temporária até o final da expiração da regra anterior, tornando possível que, após a mitigação do ataque, alguns fluxos ainda sejam redirecionados ao atacante. Esse comportamento é atenuado uma vez que o tempo de detecção é rápido o suficiente para, em geral, impedir que até mesmo o fluxo em direção ao atacante seja inserido no *switch*.

Após a realização desse tipo de mitigação, o plano de controle reflete a real rota para a vítima, sob sua visão interna e, consequentemente, os pacotes com destino aos prefixos atacados, desde que sejam encaminhados através do *AS-Defender*, são redirecionados ao cliente novamente. As setas têm o intuito de apenas indicar o sentido das mensagens *UPDATE* enviadas pela vítima e atacante sendo propagadas para algumas partes topologia.

4.6.2 A mitigação com anúncio

Em uma de suas formas de atuação, ao identificar um ataque, o *AS-Defender* anuncia prefixos com a intenção de interferir na tabela de encaminhamento dos demais ASes e atrair para si o tráfego com destino ao atacante. Ao receber o tráfego, o mesmo é redirecionado ao cliente pelo mesmo mecanismo descrito na sessão anterior.

Além dos valores em *NEXTHOP* e *ANNOUNCE*, na mitigação com anúncio, a *thread* de monitoramento insere o prefixo que deverá ser anunciado na área compartilhada, os dados são coletados e é disparado um anúncio através de todas as portas para combater o ataque.

Assim que um ataque no prefixo protegido é identificado, o mecanismo de mitigação busca a lista de subprefixos configurados no cliente afetado e realiza o anúncio de cada um deles. A mitigação funciona com o objetivo de, em caso de ataque, buscar garantir a comunicação a todos os subprefixos do cliente e, por isso, não há uma verificação sobre quais deles foram realmente afetados. A Figura 4.8 mostra o funcionamento da mitigação com anúncio.

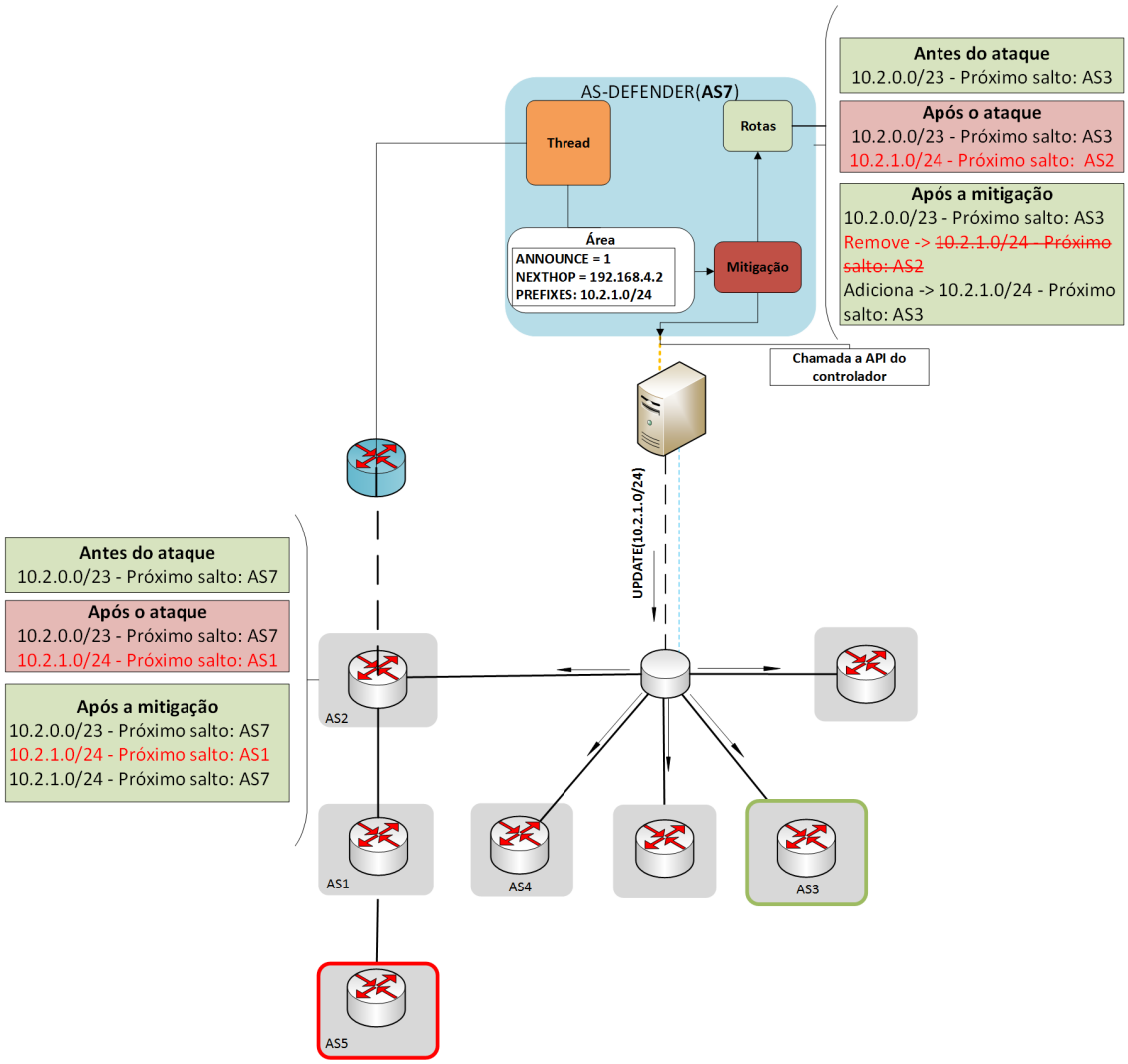


Figura 4.8: Funcionamento da mitigação com anúncio

Apenas o anúncio de mitigação foi mantido no diagrama para evitar o excesso de informações porém a sequência de anúncios segue a mesma da Figura 4.7 com a diferença do AS atacante. Neste exemplo o AS5 realiza um anúncio do subprefixo 10.2.1.0/24 e temporariamente obtém o tráfego destinado a este prefixo. Ao identificar o ataque, o mesmo prefixo é anunciado, influenciando a tabela de encaminhamento dos ASes vizinhos.

O comportamento esperado é mostrado na tabela de encaminhamento do AS2 que, logo após o ataque, possui uma entrada indicando o AS1 como o próximo salto para o prefixo 10.2.1.0/24 porém, após o mecanismo de mitigação, insere uma nova rota para o mesmo prefixo apontando para o AS7. Esse simples fato não garante que o caminho para o cliente seja estabelecido, uma vez que o processo de seleção do BGP é quem determinará o caminho a ser seguido, e este, é suficientemente complexo, podendo ser influenciado por políticas, para não ser determinado com

base apenas em trechos da tabela de encaminhamento.

4.7 Trabalhos relacionados

Através de um extensa pesquisa na literatura atual sobre o *prefix hijacking*, sua detecção e mitigação, foram elencados os trabalhos com maior relação com este, mesmo que em aspectos mínimos. Nesta seção, serão abordadas algumas das propostas que possuem uma similaridade de objetivos com o presente trabalho, contudo, em termos de funcionamento e avaliação, as soluções costumam ter específicas peculiaridades.

4.7.1 HEAP: Avaliação confiável de ataques *BGP hijacking*

O HEAP [73] é um programa de alerta para eventos de sequestro de prefixos. Ele não pretende ser um sistema de detecção de ataques mas sim auxiliar os sistemas previamente existentes a reduzir as suas taxas de alarmes falsos. Para isso, este estudo, cria um modelo formal de roteamento na internet onde é possível realizar a classificação de eventos de sequestro de prefixos, além do seu impacto e detectabilidade.

A base de raciocínio para a detecção segue a premissa de que, embora o atacante possa sequestrar um prefixo no BGP, este não tem capacidade de alterar as fontes de dados relacionadas a operação dessas redes sequestradas. Nesse caso, é possível descartar um ataque se as fontes consultadas legitimarem uma situação suspeita.

A determinação sobre um evento de sequestro de prefixo é feita através de três fontes principais: Os registros de roteamento na internet, o uso de um algoritmo de inspeção em topologias e varreduras regulares de protocolos SSL/TLS na Internet. A Figura 4.9 mostra uma visão geral da arquitetura HEAP.

As entradas externas são fornecidas por outros sistemas de detecção para que sejam avaliadas dentro da metodologia de filtragem. Os registros de roteamento na internet são utilizados para inferir relacionamentos legítimos entre um possível atacantes e a sua vítima. Adicionalmente, é realizada uma análise na topologia para saber se ocorreram mudanças decorrentes de comportamentos benignos provenientes de práticas comumente operacionais. Por fim é realizada uma varredura na internet para identificar os *hosts* habilitados para o protocolo SSL/TLS e realizar a comparação das chaves públicas antes e durante o evento.

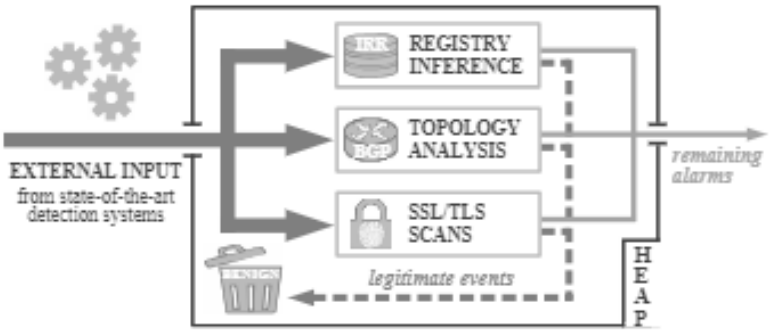


Figura 4.9: Arquitetura geral do HEAP

As avaliações foram realizadas com o uso de dados de tabelas BGP exportadas com os prefixos existentes no sistema de roteamento global e mostraram a eficácia da abordagem. As principais contribuições elencadas pelo trabalho foram: a formalização de comportamentos no roteamento global através de linguagens formais e o esquema para avaliar a validade dos alarmes de sequestro de prefixo emitidos pelos sistemas de detecção, que é preformado pelo HEAP.

4.7.2 PHAS: Um sistema de alerta para o *prefix hijacking*

O PHAS [39] é um sistema de notificação, em tempo real, que permite proprietários de prefixos detectarem o ataque *prefix hijacking* em seus IP´s e tomarem uma ação para resolver o problema.

A abordagem utilizada examina dados de roteamento BGP coletados do *Route Views*, RIPE ou qualquer outro coletor de rotas na internet. Os proprietários de prefixos que estejam interessados em receber alertas precisam se registrar no servidor PHAS e informar seus contatos de e-mail.

Através dos dados de monitoramento BGP, o PHAS mantém um conjunto de origens para um determinado prefixo e, caso uma mudança ocorra nesse conjunto, um evento é gerado. Após a geração de um evento o sistema decide se a ocorrência deve ser efetivada em uma mensagem para o proprietário.

Um filtro local para processamento de notificações pode ser utilizado pelo usuário, afim de tornar o sistema mais amigável. Esse filtro permite o gerenciamento dos e-mails externos cadastrados pelo proprietário dos prefixos e também fornece uma forma de se ajustar o conjunto de origens válidas, reportando um alarme ao administrador apenas quando ocorre uma mudança de origem inesperada. A Figura 4.10 mostra a arquitetura do sistema PHAS.

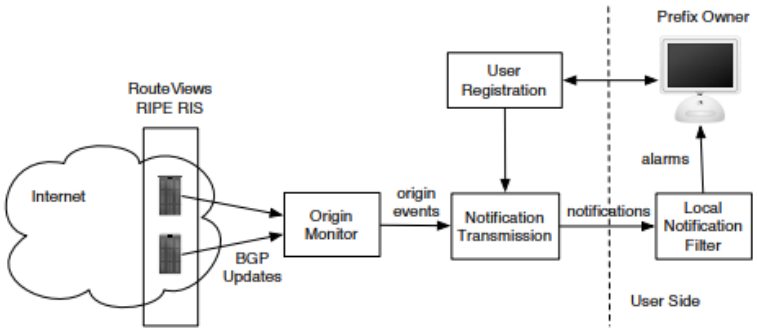


Figura 4.10: Arquitetura do sistema PHAS

O fluxo de funcionamento segue o sentido das setas onde inicialmente os dados são recebidos a partir dos coletores de rotas na internet, em seguida o monitor de origens verifica se o prefixo teve alguma origem inesperada e encaminha o evento ao transmissor de notificações que por sua vez entrega ao filtro local, definido pelo usuário do sistema. Finalmente o alarme é gerado e enviado ao proprietário.

Para realizar as avaliações foram utilizados dados de *logs* do BGP para calcular o número de eventos gerados pelo servidor do PHAS. Além disso, o mecanismo foi aplicado a dados coletados durante eventos conhecidos para verificar a eficácia do sistema.

4.7.3 DARSHANA: Detectando sequestro de rotas para confidencialidade na comunicação

O projeto DARSHANA [17] é uma solução de monitoramento que detecta o sequestro de rotas BGP através de informações do plano de dados e possui mecanismos de redundância que possibilitam a detecção, mesmo em caso de contramedidas adotadas pelo atacante.

O monitoramento do ataque envolve a análise da latência de rede (chamado de Lat), a estimativa de contagem de saltos (chamado de Hop), o calculo da similaridade de caminhos (chamado de *Path*) e a verificação do *delay* de propagação (chamado de Prop). Essa atividades são realizadas por diversos componentes que, juntos trabalham para chegar a um parecer sobre a existência de um ataque. A Figura 4.11 mostra um fluxograma que representa o seu funcionamento.

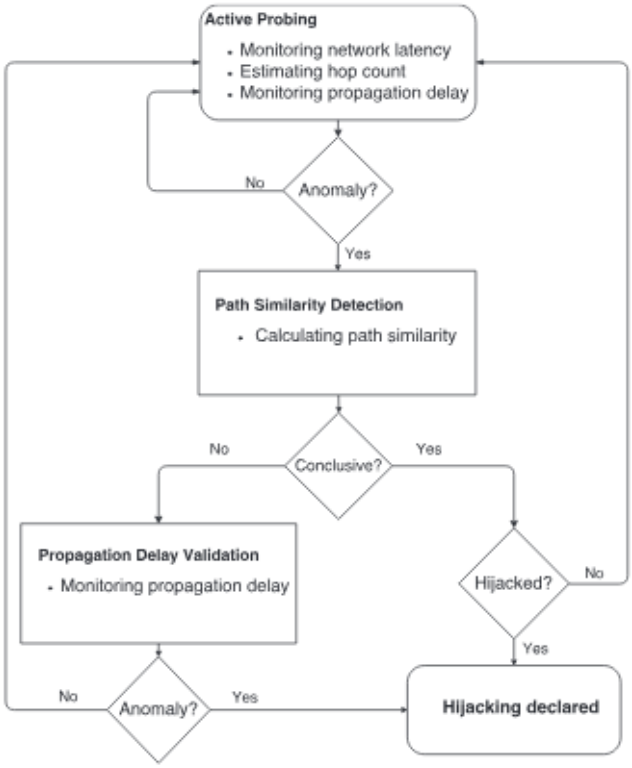


Figura 4.11: Fluxograma do DARSHANA

O componente de sondagem ativa realiza os mecanismos Lat, Hop e Prop, onde o sistema constantemente verifica os valores do RTT, a contagem de saltos e o caminho que os pacotes estão realizando. Após encontrar uma anomalia através do RTT, o sistema passa a utilizar a estimativa de contagem de saltos, considerada mais confiável para a aferição da anomalia. A sondagem ativa também executa a primeira fase da verificação do *delay* de propagação.

Em seguida, caso uma anomalia seja identificada, o sistema verifica o curso que os dados estão tomando através do comando *traceroute* para calcular a similaridade dos caminhos. A discordância de caminhos acima de um limiar definido é considerada um indicativo da ocorrência de um ataque. Caso não sejam obtidos resultados conclusivos na verificação dos caminhos, o sistema realiza a segunda fase do *delay* de propagação e, havendo persistência de dados anômalos, um ataque é declarado.

As validações do funcionamento do DARSHANA foram realizadas através de simulações de ataques utilizando máquinas virtuais hospedadas no ambiente de computação da *Amazon AWS* e no *PlanetLab*. Para isso, foram necessários três nós: o primeiro era o responsável pela fonte de tráfego, o segundo representou o destino dos pacotes e o terceiro fazendo o papel do nó que realizava o sequestro do tráfego

trocado entre os demais. Eles foram dispostos em diferentes localizações que a *Amazon* e o *PlanetLab* possuíam infraestrutura, onde foram realizados tanto testes de detecção de ataques, performance e do número de vezes que o DARSHANA precisou recorrer a técnicas com maior *overhead* para identificar o sequestro.

4.7.4 ARTEMIS: Neutralizando o BGP *hijacking* em um minuto

O ARTEMIS [20] é uma proposta que visa a detecção do ataque BGP *hijacking*[3] para a proteção do próprio sistema autônomo. Ele realiza a detecção do ataque através do monitoramento das mensagens *UPDATE* do BGP a partir de fontes de dados públicos de roteamento disponibilizadas pelos projetos *Route Views* [21], RIPE [22] e BGPmon [40].

O projeto se conecta às fontes de dados e, a medida que recebe os anúncios, analisa posição a posição do campo *AS Path* para validar se todo o caminho de ASes é válido. Essa verificação tem como base um arquivo de configuração com o número do AS dono do prefixo e os seus vizinho. Esse arquivo possibilita somente a identificação de ataque nas duas primeira posições do campo e, para detectar divergências nas posições subsequentes, são utilizadas técnicas mais avançadas que, apesar de detectar o sequestro, podem gerar falsos positivos.

Além da detecção, o ARTEMIS realiza a mitigação do ataque de forma automática através da desagregação do prefixo atacado para posterior anúncio. Uma outra forma de implementação adotada pela proposta é a possibilidade da terceirização do processo de mitigação onde, ao detectar o ataque, o ARTEMIS envia uma mensagem a uma organização responsável pela mitigação, que anuncia o prefixo atacado para atrair uma parte do tráfego, criar um túnel e reencaminha-lo novamente ao proprietário.

O trabalho também propõe uma taxonomia descrevendo diversos tipos e cenários de ataque, que são validados na avaliação da ferramenta.

O protótipo criado interage com os o RIPE através da sua API *socket.io* e com o *BGPmon* utilizando *telnet*[4]. Através deles o sistema recebe fluxos de anúncios BGP formatados em texto plano ou XML que são filtrados para que apenas os anúncios relacionados ao prefixo protegido sejam processados.

[3] O *BGP hijacking* é também conhecido sob o termo *prefix hijacking*, largamente utilizado neste trabalho.

[4] É um protocolo de que permite a conexão à computadores remotos sobre uma rede TCP/IP

Para a validação experimental do protótipo, o trabalho utiliza a plataforma PEERING [74], uma ferramenta de *testbed* BGP para comunicação em ambiente real com ASes de diversos pontos do mundo interconectados. Foram criados três ASes representando a vítima, o AS sequestrador e o AS para a terceirização da mitigação onde diversos cenários e tipos de ataques foram avaliados para testar os mecanismos de detecção e mitigação.

4.7.5 Detectando sequestros de prefixo na Internet com Argus

A proposta do Argus é entregar um sistema ágil que detecte com precisão o *prefix hijacking*, além de rapidamente deduzir a causa das anomalias de rota. Enquanto as propostas geralmente utilizam as informações do plano de controle ou do plano de dados para buscar informações sobre o ataque, o Argus faz o uso de um mecanismo que os desenvolvedores do trabalho chamam de correlação, onde tanto as informações do plano de controle e dados são utilizadas na identificação do ataque.

A proposta considera que a relação entre as informações do plano de controle e de dados em diferentes ASes funcionam como uma espécie de impressão digital e, a partir dela, é possível descobrir se uma mudança de rota foi causada por um ataque.

São consideradas três tipos de anomalias: anomalias de origem (OA), anomalias de adjacência (AA) e anomalias de política (PA). As anomalias de origem ocorre quando o AS de origem no *AS Path* para um determinado prefixo muda, ou um novo prefixo aparece. Um ataque também pode ser causado pela alteração de alguns segmentos do *As Path* através do uso de técnicas como o *as-path prepend* ou pela remoção de alguns ASes para encurtar a largura do caminho. Essas mudanças geram as chamadas anomalias de adjacência. Já as anomalias de política são identificadas pelo Argus quando as políticas de diversos ASes impedem que uma determinada sequência seja formada no *AS Path* em direção a um prefixo e, mesmo assim, tal fato é detectado.

O Argus é composto de três módulos principais: o módulo de monitoramento de anomalias (AMM), o módulo de recuperação de IP em tempo real (LRM) e o módulo de identificação de sequestro (HIM). Uma vez que o AMM detecta uma anomalia, ele notifica o IHM para identificar se um sequestro realmente acontece. Se não for o caso, ele adicionará o novo AS de origem ou o par/tripla de ASes no banco de dados local. O LRM mantém um banco de dados com endereços de IP ativos que sejam candidatos a serem alcançáveis em cada prefixos anunciado na internet. Quando uma anomalia é detectada, o LRM tenta alcançar um IP ativo no prefixo atacado,

sendo o resultado dessa tentativa um fator chave para determinar se o prefixo alvo foi sequestrado. O banco de dados de IP´s ativos é obtido através da coleta do histórico de *traceroutes* da plataforma Archipelago [75], mantida pelo CAIDA. A Figura 4.12 mostra a arquitetura do Argus.

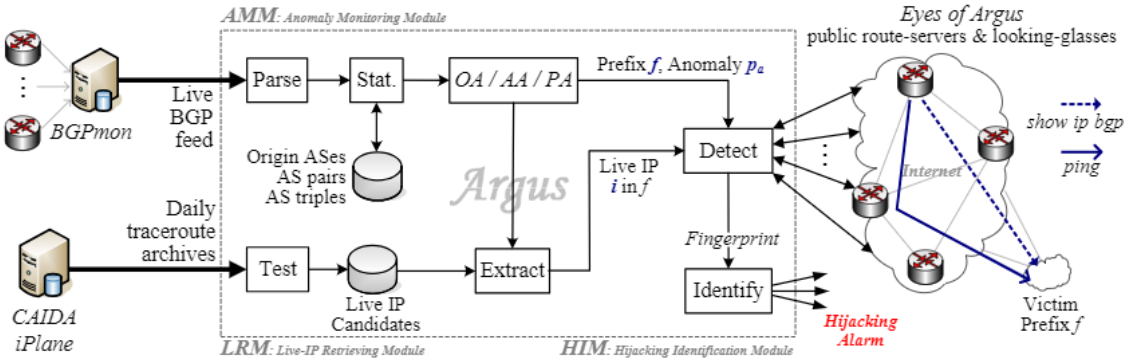


Figura 4.12: Arquitetura do sistema Argus

O HIM utiliza servidores de *looking glass*[5] na internet chamados de "Olhos do Argus". Esses servidores são utilizados para determinar se uma anomalia detectada pelo AMM é um *prefix hijacking* ou não. A partir de cada "olho", o Argus inicia dois processos, um para obter as informações de rotas (plano de controle) para alcançar a vítima e a outra para obter alcançabilidade para um IP ativo contido na sua rede através de *ping* e *traceroute*.

O Argus monitorou continuamente a internet durante um ano a partir de 2 de maio de 2011. Durante esse período diversos eventos anômalos foram identificados e reportados e, para demonstrar as vantagens do Argus, os pesquisadores realizaram uma comparação com alguns sistemas de detecção propostos anteriormente.

4.7.6 Comparação entre o *AS-Defender* trabalhos relacionados

Embora todos os trabalhos aqui relacionados tenham como objetivo propôr soluções voltadas ao *prefix hijacking*, cada um deles possui as suas particularidades. Essas diferenças vão desde as fontes de detecção até questões relacionadas à escalabilidade dos sistemas.

A Tabela 4.4 exibe uma comparação entre os principais aspectos de cada projeto, incluindo o *AS-Defender*.

[5]São uma fonte em tempo real de informações relacionadas a BGP e roteamento. Eles fornecem diversos recursos para engenheiros/pesquisadores obterem dados sobre o roteamento e até mesmo para realizar testes de rotas ou conectividade para um prefixo [76]

Projeto	**Fonte de detecção**	**Formas de mitigação**	**ASes mitigados**
HEAP	Outros sistemas	Não possui	Não possui
PHAS	Plano de controle	Não possui	Não possui
DARSHANA	Plano de dados	Não possui	Não possui
ARTEMIS	Plano de controle	Anúncio de subprefixos Terceirização de mitigação	O próprio
ARGUS	Correlação	Não possui	Não possui
AS-DEFENDER	Plano de controle	Anúncio de subprefixos Redirecionamento de tráfego	Todos os vizinhos

Tabela 4.4: Comparação entre trabalhos relacionados e o *AS-Defender*

Em relação às fontes de detecção, é possível notar diferenças entre os trabalhos, tornando notório os esforços a partir de diferentes recursos para detectar o ataque. Quanto à mitigação, apenas os projetos ARTEMIS e *AS-Defender* possuem essa capacidade. Enquanto o ARTEMIS possui a capacidade de mitigar apenas o AS onde este está atuando, o *AS-Defender* protege mais de um AS, conforme a quantidade de vizinhos habilitados para serem protegidos. Além disso, suas formas de mitigação diferem em função da necessidade do *AS-Defender* ainda realizar uma segunda etapa de redirecionamento dos dados ao cliente.

4.8 Síntese

Este capítulo elucidou a proposta deste trabalho e as suas respectivas particularidades em uma visão conceitual, sem e entrar em detalhes sobre a implantação da ferramenta, os componentes utilizados, as configurações realizadas e todo o arcabouço que possibilitou o seu funcionamento.

Foi explicada a arquitetura do *AS-Defender* e como ocorre o seu funcionamento desde a instanciação até a mitigação. A detecção ocorre com base nas informações de anúncios BGP recebidas do coletor de rotas enquanto a *thread* de monitoramento e a rotina de mitigação exercem um papel fundamental em todo o ciclo de ação contra o ataque. O arquivo de configuração de clientes fornece a base para que se saiba quais prefixos monitorar e com quais informações comparar os dados do anúncio.

A Arquitetura SDN possibilita um mecanismo de ação centralizado e baseado em programabilidade, sem a necessidade de alterar o algoritmo do BGP mas sim

realizar ações paralelas ao seu funcionamento que auxiliem um AS a responder aos eventos de sequestro.

Por fim, foram elencados alguns trabalhos que possuem relação com esse, mesmo que os meios utilizados para alcançar o objetivo não sejam exatamente iguais. Por exemplo, o fato de alguns trabalhos utilizarem dados BGP públicos disponibilizados na internet é diferente no que tange a implementação do *AS-Defender*, já que este foi implementado em um ambiente controlado, porém os objetivos relacionados ao combate do *prefix hijacking* são bastante aderentes.

5. Implantação

Este capítulo explana os aspectos técnicos sobre a implantação e as configurações dos componentes que habilitaram o funcionamento do *AS-Defender*. Serão elucidados os motivos para a escolha das tecnologias utilizadas e algumas decisões que foram tomadas para assegurar o cumprimento dos objetivos do projeto. A abordagem utilizada iniciará pela descrição do ambiente computacional, passando pelas tecnologias que tornaram possível o funcionamento do protótipo e em seguida descrevendo os demais componentes envolvidos, bem como a ferramenta utilizada para a criação da topologia.

5.1 Ambiente computacional

Para a prova de conceito, o ambiente experimental foi implementado e testado em uma máquina virtual no ambiente de computação em nuvem do *Google*, chamado *Google Cloud Platform*. A utilização dessa plataforma se deu devido a facilidade de utilizar e acessar o ambiente de qualquer lugar através de uma conexão remota e também pelo fato de ser possível customizar a máquina virtual de forma a aumentar ou diminuir os seus recursos computacionais conforme necessário.

A máquina virtual executava o sistema operacional Ubuntu 16.04.4 x86_64 LTS com a interface XFCE[1], que proporciona um ambiente gráfico com baixo consumo de recursos. Para o *hardware* virtual foram alocados duas vCPUs (processadores virtuais) na plataforma *Intel Skylake* além de 6GB de memória RAM e 20GB de disco rígido.

Embora não tenham sido utilizados efetivamente no experimento, os recursos de IP e conectividade com a internet foram fornecidos pela infraestrutura do *Google*,

[1]Disponível em https://xfce.org/

o que possibilitou a instalação e a atualização dos diversos componentes utilizados conforme necessário.

5.2 Configurações e versões dos componentes

O *AS-Defender* e todos os seus recursos e rotinas são inteiramente desenvolvidos em Python, na versão 2.7. O Python[2] é uma linguagem de programação de alto nível[3], multi-paradigma, de tipagem dinâmica[4] e que possui uma sintaxe clara e altamente legível.

A solução é composta de dois arquivos *.py*, extensão característica do Python. O primeiro, as_defender.py, é o módulo principal onde o processo BGP, a *thread* de monitoramento, a área compartilhada e os recursos utilizados do controlador são criados e executados. O segundo módulo é chamado de lib.py e é importado pelo código principal, sendo responsável pelo gerenciamento das informações do plano de controle referentes ao *switch*, como por exemplo a tabela de encaminhamento, informações das interfaces de rede e métodos que realizam a busca, adição e remoção de rotas para um destino.

No módulo lib, para executar as atividades de controle, foram importados três outros módulos: ipaddress, yaml e sys. O módulo ipaddress [77], em sua versão 1.0.19, forneceu a capacidade de inspeção e manipulação de endereços IP a partir de strings e foi necessário para popular as rotas na tabela de encaminhamento a partir de strings lidas do arquivo input.yaml. Além disso, para verificar se a tabela de encaminhamento possuía rota para um destino, é performada a operação de comparação de IP´s, também suportada por este módulo. O yaml, obtido através do pacote PyYAML [78], é responsável por fornecer rotinas para a manipulação de arquivos YAML e foi útil para a leitura e obtenção dos dados das interfaces, portas e endereços MAC do *AS-Defender*. O módulo sys [79] fornece formas de interagir diretamente com o interpretador ou acessar algumas variáveis usadas ou mantidas por ele e neste projeto foi usado simplesmente para interromper o sistema no caso de ocorrer uma exceção ao tentar acessar os dados do arquivo input.yaml.

No módulo principal foram utilizados diversos recursos da API do controlador

[2]Disponível em https://www.python.org/

[3]São linguagens de programação com um elevado nível de abstração, longe da linguagem de máquina e mais próximo à linguagem humana

[4]O tipo das variáveis é definido em tempo de execução de acordo com o valor assumido, sem a necessidade do programador explicitamente indica-lo.

que tornaram disponíveis as funcionalidades necessárias ao funcionamento do *AS-Defender*, como a manipulação de pacotes IP e ARP, a execução do BGP, a comunicação com *switch* através do protocolo *OpenFlow* e a captura de eventos de *PacketIn*. De forma complementar, alguns módulos foram importantes para viabilizar desde a conexão ao coletor de rotas até o processo de mitigação.

O módulo *threading* [80] foi crucial para a criação de uma *Thread* de execução do monitoramento, paralelamente ao funcionamento do BGP e das demais atividades da aplicação, habilitando o monitoramento dos anúncios ao mesmo tempo em que a comunicação com os ASes vizinhos era realizada. Para atender à conexão ao coletor de rotas, o pacote python-socketio [81] foi importante ao possibilitar a criação de uma conexão *WebSocket* e o recebimento das mensagens dos anúncios BGP na topologia. É importante ressaltar que não foi possível utilizar a sua versão atual, dado que esta possuía um *bug* que impactava na abertura de conexão com o coletor de rotas. Para solucionar este problema, foi empregada a versão 1.9.0.

A área de compartilhada entre a *thread* de monitoramento e o controlador foi possível através do pacote redis [82], que possibilita a criação de um mecanismo de intercomunicação adotando do paradigma de troca de mensagens publicação/-subscrição[5]. Não menos importantes, os módulos netaddr [83] e json auxiliaram na verificação do prefixo anunciado pelo atacante e na obtenção dos dados de configurações do clientes a partir do arquivo *as-defender.json* respectivamente.

Conforme brevemente mencionado na Seção 4.1, o *Ryu* [84], em sua versão 4.26, foi o controlador utilizado neste trabalho. Dentre os fatores que motivaram a sua escolha, estão o fato dele ser desenvolvido em Python e possuir em sua API recursos para a implementação de um processo BGP, além de dispor de uma boa documentação e uma lista de e-mails para postagem de dúvidas.

O coletor de rotas foi implantado com o uso do software *ExaBGP* [85] através do projeto *ExaBGP-Monitor* [86] existente no *GitHub*. Esse projeto implementa o servidor *WebSocket* fornecendo uma forma conveniente de transformar as mensagens *UPDATE* do BGP em um formato JSON para que possam ser capturadas pela *thread* de monitoramento. Ele provê um contêiner[6] baseado em *Docker*[7] com o sistema

[5]Paradigma onde o remetente não é programado para enviar uma mensagem diretamente ao destinatário, ao invés disso, a mensagem é publicada em um canal, sem o conhecimento de qual destinatário se conectará para coletar os dados enviados.

[6]São uma abstração na camada de aplicação que possibilita a execução de várias aplicações de forma totalmente isolada, em processos separados e compartilhando o mesmo *kernel* do sistema operacional.

[7]Uma ferramenta de código aberto projetada para criar e executar aplicações através de con-

operacional Alpine Linux[8].

Para a comunicação entre o *switch* e o controlador, foi utilizada a versão 1.3 do *OpenFlow* que, além de ter sido suficiente para o funcionamento da solução, é a implementação padrão adotada pelo software *Open vSwitch*[9] em sua versão 2.5.4[10], utilizada para a implantação do *switch* neste trabalho.

O *BIRD* [31] foi a solução adotada para a configuração de cada um dos ASes na topologia. O *BIRD* é um *software* para roteamento de pacotes IP em sistemas operacionais baseados em Unix. Ele é compatível com diversos protocolos de roteamento, incluindo o BGP, o que possibilitou configurar cada AS como um roteador que realiza sessões BGP com os seus vizinhos para trocar as informações sobre as rotas para cada prefixo.

Para a construção da topologia foi utilizada a ferramenta *Knet* [87], que é um *software* de código aberto responsável por criar uma topologia virtual baseada em contêineres com a tecnologia *Docker*, possibilitando cada componente ter o seu próprio sistema de arquivos, interfaces de rede e representar um processo diferente no sistema operacional. Isso tornou possível operar cada AS como uma estrutura isolada dos demais, evitando conflitos na execução de diversos processos BGP na mesma máquina. A criação da topologia é feita através de um arquivo YAML que é preenchido com os dados de todos os nós em um formato com alta legibilidade.

Com relação a localização de cada componente, apenas o próprio *AS-Defender*, o controlador e o *switch* executam diretamente sobre o sistema operacional da máquina virtual enquanto cada AS é representado por um contêiner com o sistema operacional Alpine Linux executando o BIRD para realizar as comunicações na topologia. Toda a infraestrutura de comunicação é gerenciada pelo *Knet* que cria um comutador padrão, através do *Docker*, para representar o enlace entre cada AS.

Todo o código fonte Python gerado para a criação do protótipo está publicamente acessível através do repositório do projeto *AS-Defender*[11], criado na plataforma *BitBucket*[12].

---

têineres. A sua documentação pode ser encontrada em https://www.docker.com/

[8]Documentação disponível em https://alpinelinux.org/

[9]Disponível em https://www.openvswitch.org/. O software foi utilizado na sua versão 2.5.4.

[10]A tabela de compatibilidade com as versões do protocolo *OpenFlow* está disponível em http://docs.openvswitch.org/en/latest/faq/openflow/

[11]Disponível em https://bitbucket.org/marciviniciussantos/as-defender

[12]Disponível em https://bitbucket.org/

5.3 Síntese

Este capítulo dissertou sobre os detalhes de implementação e implantação dos diversos componentes que formam o ambiente de testes, passando pelo ambiente computacional e as suas configurações de *hardware*. Também foram descritos os componentes que viabilizaram a construção do protótipo do *AS-Defender*, bem como a justificativa para os seus usos. O objetivo desse capítulo era fornecer um subsídio técnico para auxiliar no entendimento sobre como as funcionalidades descritas no Capítulo 4 puderam ser efetivamente implementadas na prática. As informações descritas também podem ser importantes para a compreensão sobre como os resultados do trabalho foram obtidos.

6. Validação experimental

Este capítulo aborda os cenários experimentais e os resultados obtidos em diversas circunstâncias analisadas, sempre apoiado pelas informações fornecidas pela proposta e pelos detalhes de implantação da solução. Também são discutidos alguns aspectos importantes sobre os resultados, bem como elencadas as limitações do protótipo e as possíveis soluções para algumas delas.

6.1 Cenários experimentais

Para validar o protótipo, foram definidos dois cenários representando duas topologias diferentes compostas por ASes interconectados em uma comunicação interdomínios através do BGP.

O primeiro cenário é composto de sete ASes sendo seis deles diretamente conectados ao *AS-Defender*, que fornece trânsito para todos os prefixos de rede que são anunciados. Os ASes *65502*, *65503*, *65504*, *65505* e *65506* recebem trânsito através do *AS-Defender* enquanto apenas o *AS65501* não possui conexão direta com ele, mas realiza uma sessão BGP com o *AS65502*. Os *hosts h1*, *h2* e *h3* estão respectivamente conectados aos ASes *65501*, *65504* e *65503* e foram criados para que fosse possível realizar a validação dos caminhos percorridos até o interior dos ASes durante os ataques e após a mitigação dos mesmos.

É importante lembrar que o roteador em azul é o coletor de rotas, que está na topologia simplesmente para disponibilizar as informações do plano de controle conforme ocorre em projetos como o *Route Views* e o RIPE. A Figura 6.1 descreve o primeiro cenário com todos os referidos detalhes.

Este cenário foi escolhido por ser mais simples, com menos conexões e uma quantidade menor de ASes, facilitando a visualização do mecanismo, a aferição da

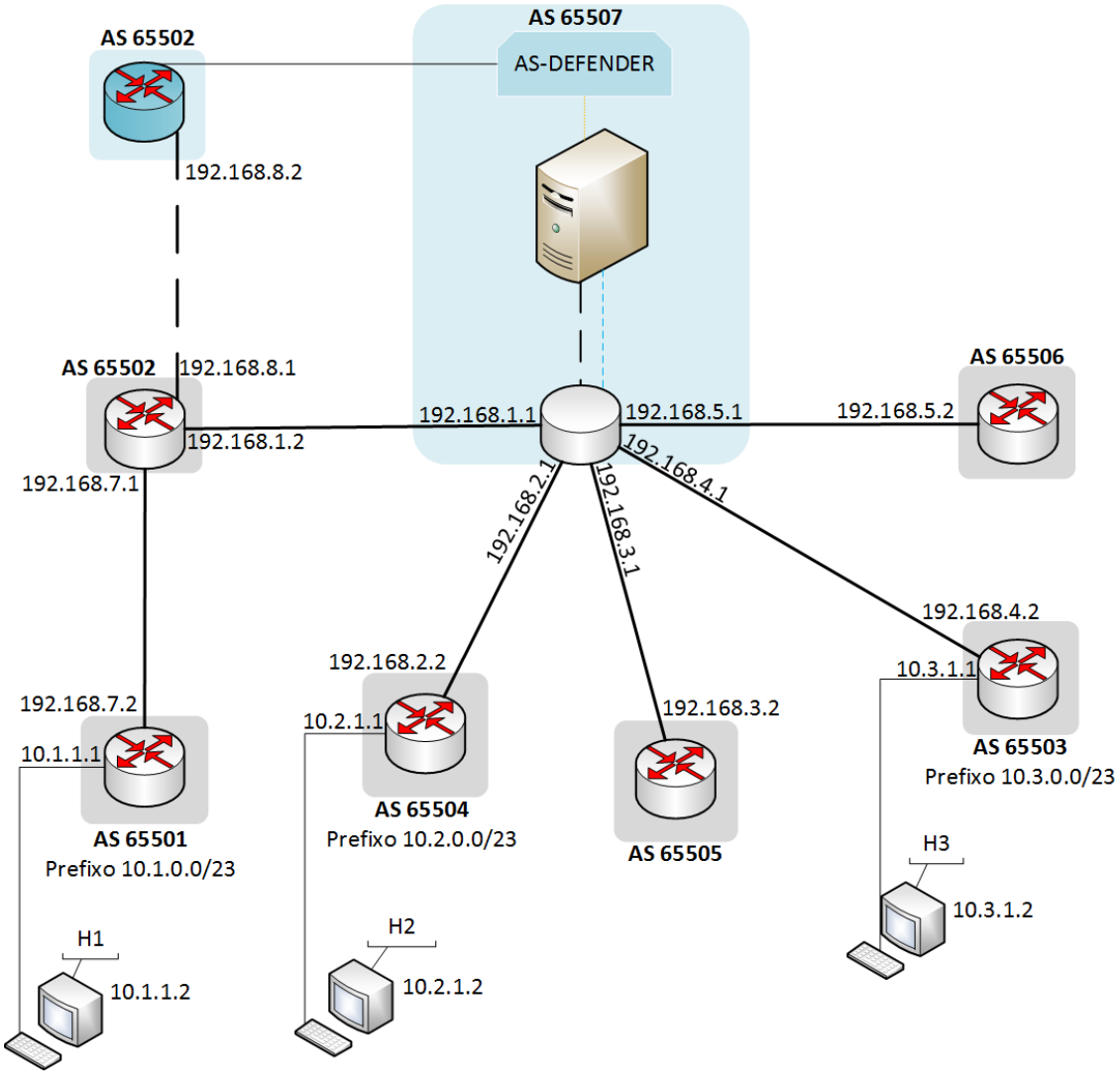


Figura 6.1: Elementos da topologia e os seus detalhes no primeiro cenário experimental

sua validade e o entendimento sobre o funcionamento da solução.

O segundo cenário é um pouco mais complexo pois é composto por um número maior de ASes, com mais conexões e, consequentemente, fornece a possibilidade de expormos o mecanismo proposto a algumas variações, para avaliarmos a validade da proposta sob algumas condições. Nele coexistem onze ASes onde, apenas um, o *AS65511*, não faz uma sessão BGP com pelo menos dois outros ASes. Vide Figura 6.2. Isso faz com que os ASes possam ter mais de uma rota para o mesmo prefixo, ajudando na realização de outras análises que se aproximem do funcionamento do BGP na internet.

Ambos os cenários foram definidos através de consultas às topologias BGP mais comuns existentes em grandes ASes da internet, como por exemplo, *AT&T* (ASN

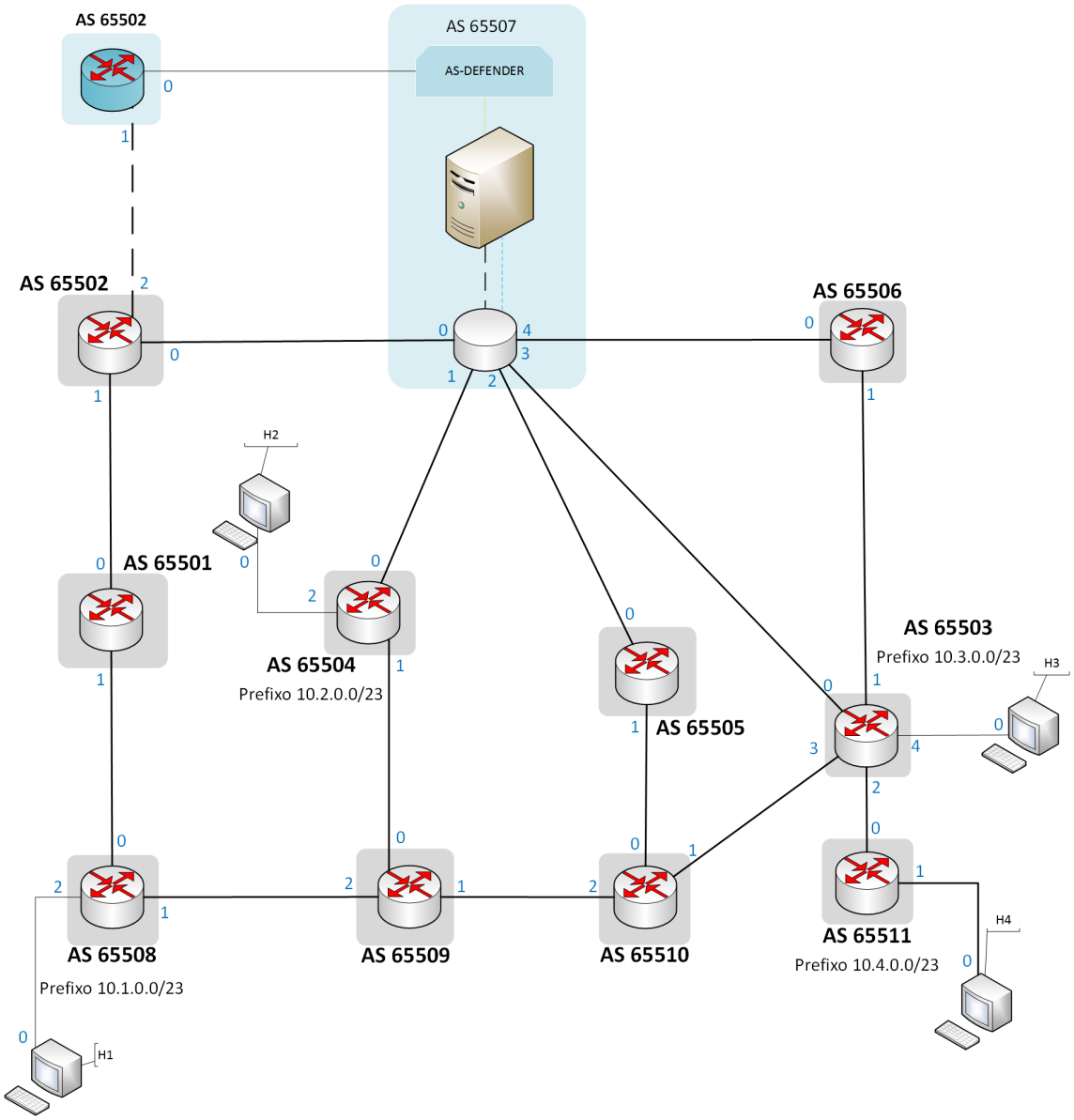


Figura 6.2: Elementos da topologia e os seus detalhes no segundo cenário experimental

7018), *Google* (ASN 15169), *Microsoft* (ASN 8075), *Facebook* (ASN 32934) e *Yahoo* (ASN 10310), por meio da ferramenta *BGPlay*[1] [88], desenvolvida pelo RIPE. Se observarmos as topologias envolvendo os ASes citados, poderemos ver uma semelhança, obviamente simplificada, com o relacionamento entre os ASes em ambos os cenários.

[1]Disponível em https://stat.ripe.net/special/bgplay

6.2 Metodologia de validação

Para que haja um bom entendimento dos resultados obtidos, é necessário que a metodologia para a validação do protótipo esteja bem definida e que os tipos de ataques, a mitigação e a política de interação entre todos o ASes da topologia estejam bem compreendidas.

Na validação do *AS-Defender* são executados os ataques de sequestro de prefixo e de subprefixos, definidos na Seção 2.4.1. Ambos os ataques são performados através do anúncio do prefixo ou subprefixo da vítima por meio de mudanças no arquivo de configuração do *BIRD* no atacante. Todos os ataques utilizados na validação são do tipo 0 (definido na Seção 2.4.1) e, embora o protótipo detecte também ataques tipo 1, os mesmos não foram utilizados pois não foram encontradas formas de induzir o atacante a modificar a segunda posição do *AS Path* por meio de configurações do BGP. Para atender o propósito de validação da detecção desse ataque, nós criamos um *script* que gerava *UPDATES* no mesmo formato JSON encaminhado pelo coletor de rotas, com a segunda posição do *AS Path* propositalmente incorreta, em relação ao arquivo de configuração do cliente. Nos nossos testes, os ataques Tipo 1 foram adequadamente detectados sem a presença de qualquer inconsistência ou falso positivo.

Como o *AS-Defender* utiliza o atributo *SUBPREFIXES* do arquivo de configuração do cliente para saber quais prefixos devem ser anunciados em caso de ataque, todas as vezes que a mitigação com anúncio foi utilizada, os prefixos configurados eram os subprefixos /24 da respectiva vítima. Isso deve ser esclarecido porque o objetivo dessa configuração é justamente anunciar os subprefixos porém, caso estes não sejam configurados adequadamente, a mitigação pode não ocorrer conforme o esperado.

Adicionalmente, para que o processo de mitigação tivesse êxito, foi necessária a configuração de um filtro de anúncios em cada AS no papel de vítima. Esse filtro se fez necessário para que o AS não importasse anúncios direcionados ao seu próprio prefixo e, consequentemente, não tentasse entregar os pacotes com destino a sua própria rede para o *AS-Defender*, após o recebimento de um anúncio durante a mitigação. Isso não representa um problema, uma vez que essa configuração poderia ser implementada em um ambiente BGP na internet sem maiores dificuldades.

Nos experimentos, não foi levada em consideração a classificação dos ataques em função da manipulação do tráfego, uma vez que a forma como o atacante tratará os

dados posteriormente ao ataque não influencia exatamente no processo de detecção ou mitigação aplicados pelo *AS-Defender*. O protótipo é avaliado em várias situações a partir da combinação entre um tipo de ataque, um tipo de mitigação, o atacante e a vítima. Essas informações são alternadas de modo que sejam criadas diversas circunstâncias diferentes e os testes sejam representativos para ocasiões que possam ocorrer em um ambiente BGP. As variações realizadas nos experimentos se referem aos seguintes aspectos:

- **Localização do atacante**: A localização do atacante é alternada de modo a avaliarmos os seus efeitos para um cenário de ataque.
- **Localização da vítima**: A localização da vítima é modificada para entendermos se há circunstâncias onde a modificação do sistema autônomo afetado influencia nos resultados do ataque.
- **Quantidade de vítimas**: Os experimentos variam entre uma ou duas vítimas para avaliarmos a aplicabilidade do *AS-Defender* em um cenário de ataque a múltiplos ASes.
- **Tipo de ataque**: São utilizados ataques de prefixo e subprefixo para identificarmos sob quais situações cada um dos ataques "polui" mais sistemas autônomos.
- **Tipo de mitigação**: As mitigações são alternadas entre com anúncio e sem anúncio, explicadas no Capítulo 4, onde são verificadas situações nas quais a solução proposta não funciona ou possui eficácia reduzida.
- **Mecanismos adicionais de defesa**: É adotado um mecanismo auxiliar de filtragem de prefixos na tentativa de melhorar a performance da mitigação sem anúncio.

O primeiro cenário tem o intuito de, basicamente, mostrar que todo o ciclo de ataque, detecção e mitigação proposto pelo *AS-Defender* funciona adequadamente, enquanto que o segundo cenário, por ser mais complexo, fornece a possibilidade da realização das diversas variações mencionadas no parágrafo anterior.

A validação da mitigação é feita por meio da tentativa de contatar o *host* que está no AS vítima utilizando o comando *traceroute*. Como não foi configurado um *host* com o mesmo IP no AS atacante, ao executar um *traceroute* a partir de um AS que foi poluído com o anuncio falso, o pacote chegará até o atacante e se perderá, pois

não há um *host* interno para a entrega do pacote, caso contrário, o contato com o *host* ocorrerá com sucesso. Em alguns casos, para constatarmos alguns comportamentos ou realizarmos testes de alcançabilidade entre os nós, foi utilizado o comando *ping*.

Para aferir os resultados adequadamente, foi imposto um *delay* proposital entre a detecção e a mitigação do ataque. Sem esse mecanismo, não seria possível verificar a diferença de comportamento entre os nós da topologia nos diferentes momentos, dado que o processo de mitigação inicia instantaneamente após a detecção.

O objetivo geral da validação experimental é, além de mostrar que a proposta funciona, verificar o comportamento do protótipo sob algumas circunstâncias, identificar em quais delas os resultados são melhores ou piores e procurar buscar alternativas para as situações menos favoráveis, seja aplicando novos testes ou sugerindo melhorias no protótipo.

Quanto a formatação dos resultados, para que estes sejam apresentados de uma forma mais elegante e facilmente compreensível, serão apresentadas tabelas mostrando informações dos ASes em alguns momentos, mostrando quais deles recuperaram o acesso à vítima após a mitigação.

Em todos os resultados, ao reproduzir as informações das tabelas de encaminhamento, os IP´s dos próximos saltos foram substituídos pelos ASN´s para facilitar o entendimento e evitarmos a necessidade de, constantemente, precisarmos consultar a topologia para validar a correspondência entre um AS e o seu respectivo IP. O mesmo ocorre quando representamos o caminho percorrido por um pacote dentro da topologia.

6.3 Resultados e discussões

Nesta seção são analisados os resultados obtidos através dos experimentos realizados com o *AS-Defender*. Para fins de organização, os dados obtidos foram separados de acordo com o cenário utilizado. Os resultados iniciais foram coletados no primeiro cenário abordando os dois tipos de ataques testados (prefixo e subprefixo) juntamente com a mitigação com e sem anúncio. Em seguida utilizamos o segundo cenário com algumas variações enunciadas na seção anterior. Ao final, sumarizamos os dados obtidos nos experimentos para avaliarmos de uma forma geral o comportamento do protótipo.

6.3.1 Primeiro cenário

Para validar as funcionalidades do protótipo nós realizamos um ataque de prefixo, enquanto o *AS-Defender* foi configurado para realizar a defesa do cliente com a mitigação com anúncio. Em seguida, fizemos um ataque de subprefixo e configuramos o protótipo para mitigar o ataque sem anúncios. Essas combinações foram propositalmente definidas de modo a combinar o tipo de ataque teoricamente menos impactante com o método de defesa mais disruptivo e vice-versa, afim de visualizarmos um contraponto entre o momento do ataque e a mitigação. Por fim, nós fazemos um ataque de subprefixo com a mitigação com anúncio para visualizar a eficácia do mecanismo em um cenário simples.

Em ambos os experimentos o *AS65501* foi definido como o atacante, enquanto o *AS65503* fazia o papel da vítima. Essa definição foi simplesmente baseada no fato do atacante ser o único AS que não é diretamente conectado ao protótipo e, portanto, ser o mais indicado para esse papel, enquanto a vítima é um dos ASes que podem ser defendidos. A Tabela 6.1 define os experimentos realizados no primeiro cenário, especificando os ASes (atacante e vítima) envolvidos e os prefixos anunciados no ataque ("P.Ataque") e na mitigação ("P.Mitigação").

Nº experimento	**Atacante**	**Vítima**	**P.Ataque**	**P.Mitigação**
1	AS 65501	AS 65503	10.3.0.0/23	10.3.0.0/24 e 10.3.1.0/24
2	AS 65501	AS 65503	10.3.1.0/24	Sem anúncio
3	AS 65501	AS 65503	10.3.1.0/24	10.3.0.0/24 e 10.3.1.0/24

Tabela 6.1: Especificação dos experimentos realizados no primeiro cenário

Para um bom entendimento da dinâmica de apresentação das informações, os resultados do primeiro experimento serão mostrados através do uso tanto de diagramas quanto tabelas, para que seja possível fazer uma comparação entre o dados e o que ocorreu na topologia. Em seguida, as demais informações ficarão dispostas apenas em tabelas afim de evitarmos a todo momento colocarmos a mesma topologia no trabalho.

As Figuras 6.3 e 6.4 mostram que, após o ataque no prefixo 10.3.0.0/23, apenas o próprio atacante e o *AS65502* foram afetados, passando a preferir a rota para o caminho falso, enquanto que, após a mitigação com anúncio, todos os ASes passaram a novamente preferir o caminho em direção a vítima.

Na Figura 6.3, o tracejado em vermelho representa o caminho percorrido pelo

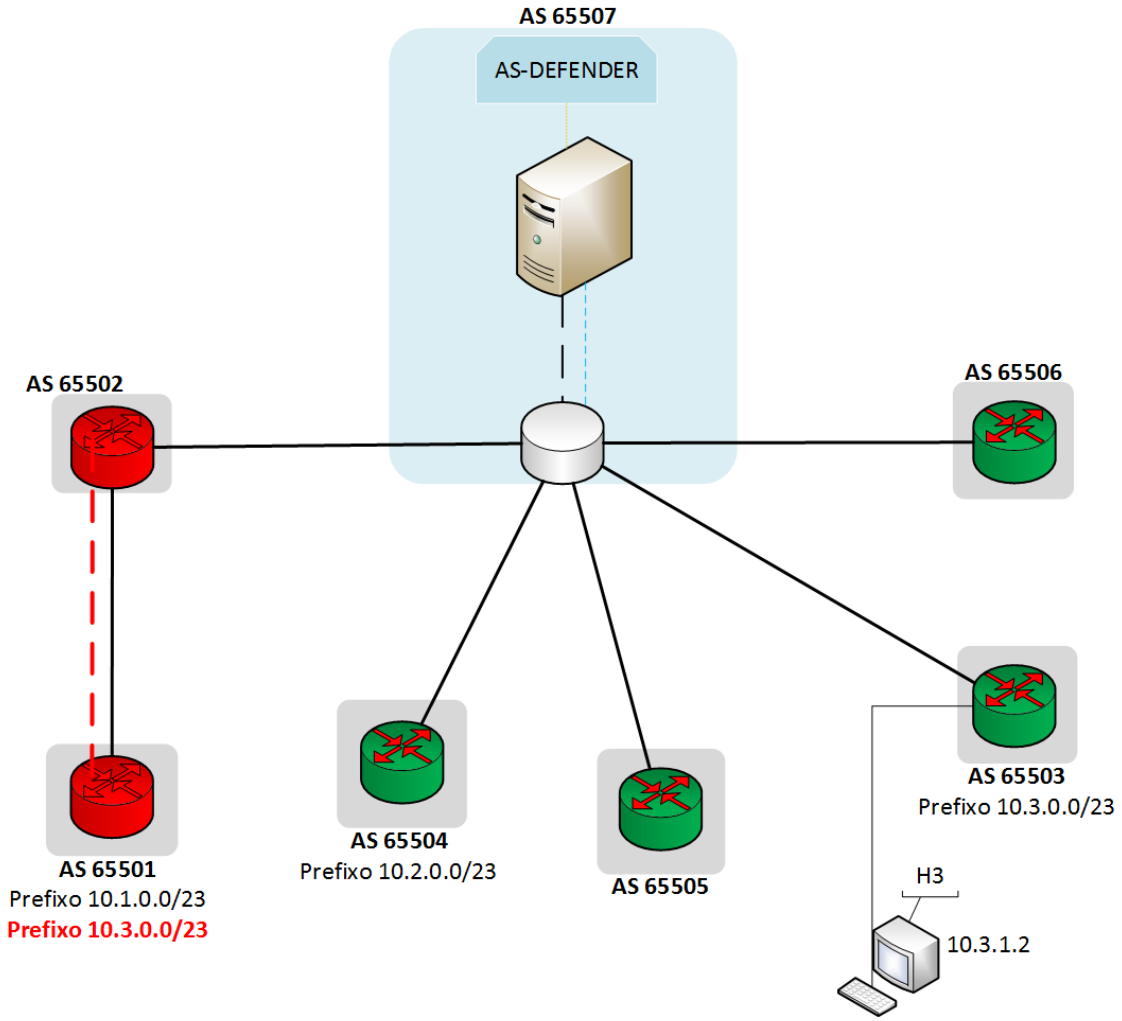


Figura 6.3: Resultado do ataque no experimento 1

AS65502 em direção ao IP 10.3.1.2 (IP do *host* interno ao *AS65503*) enquanto que, na Figura 6.4, o tracejado verde possui o mesmo significado.

Para validar se todas as fases do processo foram eficazes, foi necessário acompanhar as tabelas de encaminhamento dos ASes para saber se os anúncios do atacante e do *AS-Defender* geraram as mudanças esperadas. Além disso, a tabela de encaminhamento do próprio *AS-Defender* sofre modificações quando o processo de mitigação é acionado. Neste primeiro experimento o *AS65502* foi escolhido para acompanharmos e mostrarmos o funcionamento interno da solução. Para isso, a Tabela 6.2 mostra o que ocorreu nas tabelas de encaminhamento dos ASes *65501*, *65502* e do *AS-Defender* com relação ao prefixo atacado. Podemos ver que, durante o ataque, o estado da tabela de encaminhamento do *AS-Defender* não se alterou pois o anúncio do atacante não gerou uma melhor rota para o prefixo atacado. Em função disso, não foram geradas mudanças significativas nas entradas da tabela de fluxo do *switch*, que poderão ser vistas em outros experimentos.

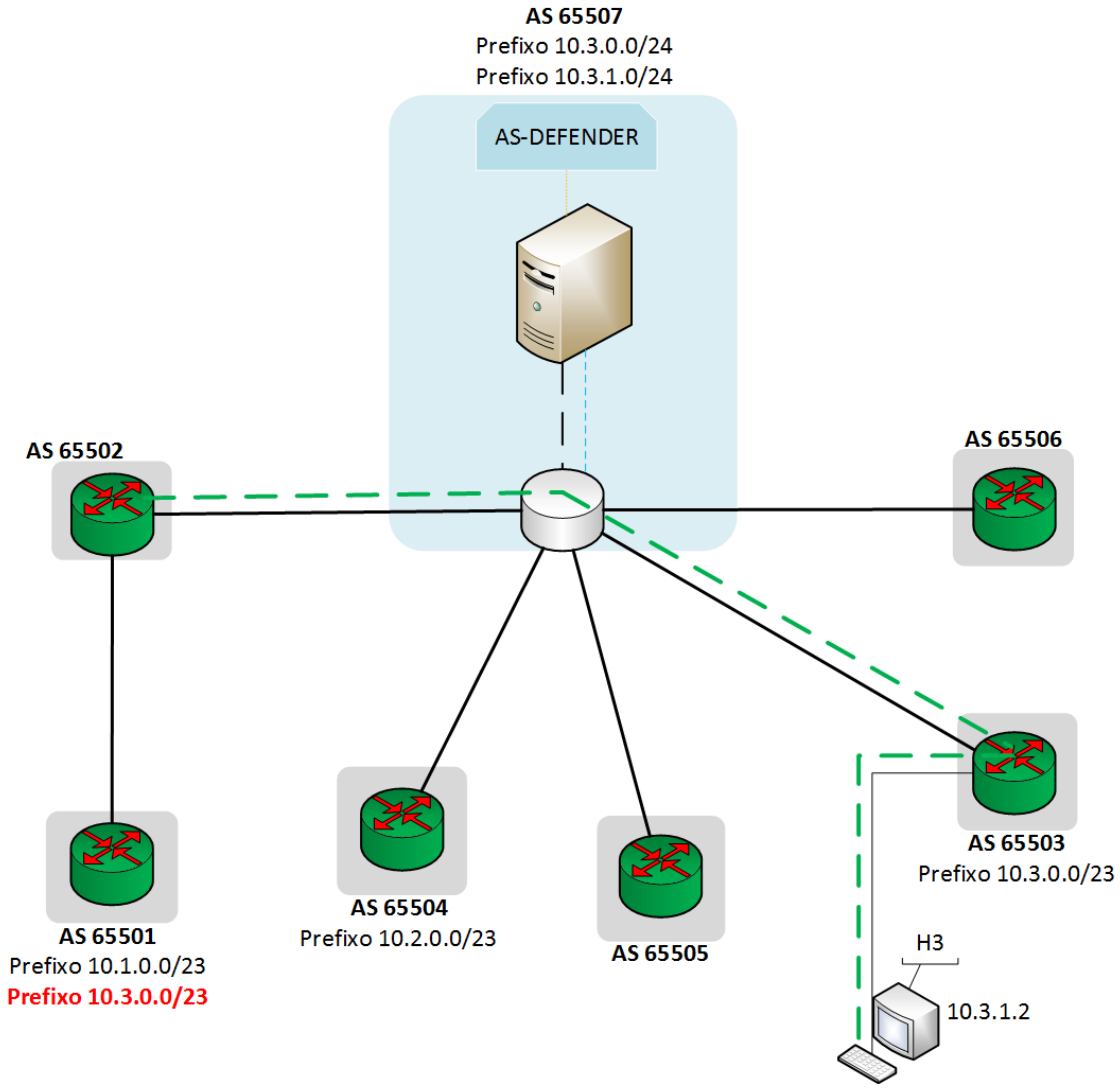


Figura 6.4: Resultado após a mitigação com anúncio no experimento 1

	Antes do Ataque	**Durante o Ataque**	**Após a mitigação**
AS-Defender	10.3.0.0/23 via AS 65503	10.3.0.0/23 via AS 65503	10.3.0.0/23 via AS 65503 10.3.0.0/24 via AS 65503 10.3.1.0/24 via AS 65503
AS 65502	10.3.0.0/23 via AS 65507	10.3.0.0/23 via AS 65501	10.3.0.0/23 via AS 65501 10.3.0.0/24 via AS 65507 10.3.1.0/24 via AS 65507
AS 65501	10.3.0.0/23 via AS 65502	10.3.0.0/23 via AS 65501	10.3.0.0/23 via AS 65501 10.3.0.0/24 via AS 65502 10.3.1.0/24 via AS 65502

Tabela 6.2: Mudanças ocorridas na tabela de encaminhamento durante o ataque no experimento 1.

Embora, de acordo com as informações da tabela 6.2, haja um forte indicativo

de que a mudança ocorrida na tabela de encaminhamento do *AS65502*, após a mitigação, tenha sido causada por um anúncio do *AS-Defender* (*AS65507*), apenas com essa informação não foi possível ter total certeza disso. Para comprovar essa premissa, a tabela de roteamento BGP do *AS65502* foi verificada para validarmos que os prefixos 10.3.0.0/24 e 10.3.1.0/24 foram efetivamente anunciados pelo protótipo. A Tabela 6.3 mostra a tabela de roteamento[2] BGP no *AS65502*, onde ficou claro que o mecanismo de anúncio funcionou corretamente.

Antes do ataque	**Durante o Ataque**	**Depois da Mitigação**
10.3.0.0/23 [AS65503i] via AS65507	10.3.0.0/23 [AS65501i] via AS65501 [AS65503i] via AS65507	10.3.0.0/24 [AS65507i] via AS65507 10.3.1.0/24 [AS65507i] via AS65507 10.3.0.0/23 [AS65501i] via AS65501 [AS65503i] via AS65507

Tabela 6.3: Mudanças ocorridas na tabela de roteamento BGP do AS65502 no experimento 1

As informações mostram que inicialmente o *AS65502* possui apenas uma rota para o prefixo alvo que havia sido anunciada pelo *AS65503* (vítima) e que tinha como próximo salto o *AS-Defender*. Após o ataque, apareceu uma nova rota para o mesmo prefixo, porém desta vez anunciada pelo atacante e que passou a ser a rota preferida. Após a mitigação, aparecem duas novas rotas que indicam que os prefixos 10.3.0.0/24 e 10.3.1.0/24 foram anunciados pelo protótipo(a informação "AS65507i" entre colchetes indica isso no software *BIRD*).

Para fornecermos uma visão geral dos efeitos do ataque e da mitigação, a cada experimento, será mostrada uma tabela com o caminho percorrido por cada AS em direção ao IP do *host* interno localizado na vítima, bem como os ASes que foram poluídos[3] com o anúncio falso e recuperados pelo *AS-Defender*. A Tabela 6.4 mostra essa visão onde CAA (Caminho antes do ataque), CDA (Caminho durante o ataque) e CDM (Caminho depois da mitigação) são respectivamente os caminhos percorridos por cada AS antes do ataque, durante o ataque e depois a mitigação. A coluna "Mitigado" possui os valores "Sim" ou "Não" conforme cada AS acessou o *host* na vitima ao final da mitigação. É possível observar que, embora alguns ASes não tenham sido poluídos, a coluna referente a mitigação está com o valor "Sim". Isso ocorre porque ela busca dar uma visão de todos os ASes que, ao fim de todo o

[2]Também conhecida como RIB, é uma tabela que contém informações de roteamento a partir de algum protocolo. Diferentemente da FIB (tabela de encaminhamento), ela contém não apenas a melhor rota para cada rede, mas todas as rotas disponíveis.

[3]Este termo foi utilizado com base em outros trabalhos científicos referenciados nessa dissertação. Ele determina os ASes que assimilaram a rota falsa anunciada pelo atacante.

ciclo de defesa, estavam acessando a vítima, tendo sido poluídos ou não.

AS	**CAA**	**CDA**	**CDM**	**Poluído**	**Mitigado**
65501	65502 - 65507 - 65503	interno	65502 - 65507 - 65503	Sim	Sim
65502	65507 - 65503	65501	65507 - 65503	Sim	Sim
65503	interno	interno	interno	Não	Sim
65504	65507 - 65503	65507 - 65503	65507 - 65503	Não	Sim
65505	65507 - 65503	65507 - 65503	65507 - 65503	Não	Sim
65506	65507 - 65503	65507 - 65503	65507 - 65503	Não	Sim
65507	65503	65503	65503	Não	Sim

Tabela 6.4: Resultados do experimento 1

De acordo com os resultados, é possível verificar que, inclusive o atacante (*AS65501*), voltou a ter acesso à vítima após a mitigação. Isso ocorreu uma vez que, ao receber o anúncio mais específico gerado pelo protótipo, o mecanismo de seleção do melhor caminho do BGP passou a preferir esta rota para acessar o IP do *host H3*.

O valor "interno" significa que, ao tentar acessar o ip do *host*, de acordo com a tabela de encaminhamento do AS, era indicado que o IP de destino pertencia ao próprio AS e, portanto, o caminho até o *host* não envolvia nenhuma rota externa. Os ASes poluídos são os que, ao receberem o anúncio do atacante, passaram a escolher a rota falsa como o melhor caminho para acessar o *host H3* enquanto que os ASes mitigados são os que acessam o *host* adequadamente após o *AS-Defender* ativar o processo de mitigação.

Conforme a Tabela 6.1, para o segundo experimento, um ataque de subprefixo foi disseminado na topologia ao mesmo tempo em que buscamos testar a efetividade da mitigação sem anúncio em uma situação na qual diversos clientes, para se comunicar uns com os outros, precisam utilizar uma rota que passa obrigatoriamente pelo *AS-Defender*.

Sob essas circunstâncias, o *AS-Defender* foi afetado pelo ataque e, até que realizasse a mitigação, esse efeito pôde ser visto nas entradas inseridas na tabela de fluxo do *switch*. A Tabela 6.5 mostra as entradas de fluxo inseridas no *switch* durante a tentativa de comunicação com o *host H3* a partir do *AS65504*.

	Prioridade	Correspondência	Ação
Antes do ataque	1	ip,in_port=2,nw_src=192.168.2.2,nw_dst=10.3.1.2	output:4
	1	in_port=4,nw_src=10.3.1.2,nw_dst=192.168.2.2	output:2
Durante o ataque	1	ip,in_port=2,nw_src=192.168.2.2,nw_dst=10.3.1.2	output:1
Após a mitigação	1	ip,in_port=2,nw_src=192.168.2.2,nw_dst=10.3.1.2	output:4
	1	ip,in_port=4,nw_src=10.3.1.2,nw_dst=192.168.2.2	output:2

Tabela 6.5: Entradas de fluxo inseridas no *switch* durante o experimento 2

Antes do ataque, as entradas de fluxo representavam uma comunicação normal entre os nós. Já durante o ataque, o próprio protótipo, ao receber o anúncio do atacante, determinou como melhor caminho a rota falsa e portanto, ao receber um *PacketIn*, inseriu uma entrada de fluxo apontando a saída para a porta 1, que representa o caminho para o atacante. Durante o ataque, apenas uma entrada de fluxo foi inserida, devido a não existir um *host* falso para o retorno da comunicação.

Ao mesmo tempo, a tabela 6.6 mostra que, na fase de mitigação, não foi recebido nenhum anúncio pelo *AS65504* vindo do protótipo.

Antes do ataque	Durante o ataque	Após a mitigação
10.3.0.0/23 [AS65503i] via AS65507	10.3.1.0/24 [AS65501i] via AS65507	10.3.1.0/24 [AS65501i] via AS65507
	10.3.0.0/23 [AS65503i] via AS65507	10.3.0.0/23 [AS65503i] via AS65507

Tabela 6.6: Mudanças ocorridas na tabela de roteamento BGP do AS65504 no experimento 2

Por fim, a análise do ataque de subprefixo combinado com a mitigação sem anúncio nos mostrou que além de todos os ASes, exceto a vítima, terem sido poluídos com o anúncio do atacante, apenas os ASes *65501* e *65502* não foram mitigados. É de certa forma óbvio que o próprio atacante e o seu vizinho não fossem corrigidos dado que, em um cenário real, no caso de um ataque de subprefixo com uma máscara /24, o anúncio da mitigação não poderia ser mais específico do que isso, devido a filtragens impostas pelos roteadores na internet e, portanto, o atacante daria preferência a sua própria rota, bem como o vizinho que poderia preferir esse caminho em função da menor largura do *AS Path* (Vide Tabela 6.7)

AS	CAA	CDA	CDM	Poluído	Mitigado
65501	65502 - 65507 - 65503	interno	interno	Sim	Não
65502	65507 - 65503	65501	65501	Sim	Não
65503	interno	interno	interno	Não	Sim
65504	65507 - 65503	65507 - 65502 - 65501	65507 - 65503	Sim	Sim
65505	65507 - 65503	65507 - 65502 - 65501	65507 - 65503	Sim	Sim
65506	65507 - 65503	65507 - 65502 - 65501	65507 - 65503	Sim	Sim
65507	65503	65502 - 65501	65503	Sim	Sim

Tabela 6.7: Resultados do experimento 2

Para o terceiro experimento, foi determinado um ataque de subprefixo e a mitigação com anúncio para validarmos a ação de defesa, em uma situação onde o ataque fosse realizado no mesmo subprefixo da vítima que é anunciado pelo *AS-Defender* para combater o ataque. Os respectivos resultados podem ser vistos na Tabela 6.8

AS	CAA	CDA	CDM	Poluído	Mitigado
65501	65502 - 65507 - 65503	interno	interno	Sim	Não
65502	65507 - 65503	65501	65507 - 65503	Sim	Sim
65503	interno	interno	interno	Não	Sim
65504	65507 - 65503	65507 - 65502 - 65501	65507 - 65503	Sim	Sim
65505	65507 - 65503	65507 - 65502 - 65501	65507 - 65503	Sim	Sim
65506	65507 - 65503	65507 - 65502 - 65501	65507 - 65503	Sim	Sim
65507	65503	65502 - 65501	65503	Sim	Sim

Tabela 6.8: Resultados do experimento 3

Após a mitigação, somente o atacante continuou preferindo a sua rota falsa. Os demais, ao terem o *AS-Defender* no seu caminho para o prefixo atacado, foram adequadamente direcionados para o AS vítima. O *AS65502*, embora estivesse a mesma distância do atacante e do protótipo, além de ter recebido o anúncio para o mesmo prefixo dos dois, preferiu utilizar o caminho em direção à vítima. Ao analisarmos as regras de seleção de rotas do BIRD, descritas na Seção 2.3, foi verificado que o único atributo que parece não ser igual entre os anúncios é o *router-id* e portanto esse pode ter sido o fator determinante para a escolha do caminho correto.

6.3.2 Segundo cenário

No primeiro cenário, como o *AS-Defender* está diretamente conectado a quase todos os ASes, ele se encontra em uma posição privilegiada, estando a um salto

de quase todos os nós da topologia. Essa vantagem é um dos fatores que podem alavancar o desempenho dos mecanismos de mitigação. Além do mais, o protótipo é um salto obrigatório para quase todas as rotas, significando que os pacotes com quase todas as relações origem/destino passam por ele, o que representa mais um item relevante que pode melhorar o desempenho do mecanismo. Diante disso, o segundo cenário insere novas variáveis no roteamento, visto que novos ASes são colocados na topologia, possibilitando novas conexões e caminhos para se alcançar cada prefixo. A Tabela 6.9 ilustra, no mesmo formado da Tabela 6.1, os experimentos realizados no segundo cenário.

Nº experimento	**Atacante**	**Vítima**	**P.Ataque**	**P.Mitigação**
4	AS 65501	AS 65503	10.3.0.0/23	10.3.0.0/24 e 10.3.1.0/24
5	AS 65501	AS 65503	10.3.1.0/24	Sem acúncio
6	AS 65508	AS 65503	10.3.1.0/24	10.3.0.0/24 e 10.3.1.0/24
7	AS 65508	AS 65503	10.3.1.0/24	Sem anúncio
8	AS 65509	AS 65504 e AS 65503	10.3.1.0/24 e 10.2.1.0/24	10.3.1.0/24 e 10.2.1.0/24
9	AS 65509	AS 65504 e AS 65503	10.3.1.0/24 e 10.2.1.0/24	Sem Anúncio
10	AS 65510	AS 65503	10.3.1.0/24	10.3.0.0/24 e 10.3.1.0/24
11	AS 65503	AS 65504	10.3.1.0/24	10.3.0.0/24 e 10.3.1.0/24
12*	AS 65508	AS 65503	10.3.1.0/24	Sem anúncio

Tabela 6.9: Especificação dos experimentos realizados no segundo cenário

Os experimentos 4 e 5 são iguais aos que foram feitos no primeiro cenário e têm como objetivo a comparação dos efeitos tanto do ataque quanto da mitigação, quando a topologia é expandida e mais de uma rota para o mesmo caminho é possível. Já os demais experimentos têm o propósito de testar outras situações que não seriam interessantes, ou mesmo possíveis, no primeiro cenário. O experimento 12 está marcado com um "*" pois, apesar de ter as mesmas características do experimento 7, possui uma configuração de filtragem de anúncios BGP que tem como objetivo avaliarmos a possibilidade de melhorar o resultado da mitigação sem anúncio.

Após a realização do quarto experimento (Vide Tabela 6.10), foi possível perceber algumas diferenças em relação ao ocorrido com o primeiro cenário. Logo, em comparação com os resultados apresentados na Tabela 6.4, foi identificado que, em uma topologia maior, mais ASes foram afetados pelo ataque. Além disso, a existência da possibilidade de vários caminhos para um mesmo prefixo, fez com que alguns ASes tomassem caminhos mais longos (marcados com um *) após o processo de mitigação. Em alguns casos, como no do *AS65506*, a rota escolhida passava pelo *AS-Defender* quando na verdade, para alcançar o prefixo atacado, era suficiente utilizar a interface que está diretamente conectada à vítima. De uma forma geral,

se olharmos apenas os ASes poluídos e mitigados, o comportamento é semelhante, mas com relação aos caminhos percorridos, há algumas diferenças.

AS	CAA	CDA	CDM	Poluído	Mitigado
65501	65502 - 65507 - 65503	interno	65502 - 65507 - 65503	Sim	Sim
65502	65507 - 65503	65501	65507 - 65503	Sim	Sim
65503	interno	interno	interno	Não	Sim
65504	65507 - 65503	65507 - 65503	65507 - 65503	Não	Sim
65505	65507 - 65503	65507 - 65503	65507 - 65503	Não	Sim
65506	65503	65503	65507 - 65503*	Não	Sim
65507	65503	65503	65503	Não	Sim
65508	65509 - 65510 - 65503	65501	65501 - 65502 - 65507 - 65503*	Sim	Sim
65509	65510 - 65503	65508 - 65501	65504 - 65507 - 65503*	Sim	Sim
65510	65503	65503	65505 - 65507 - 65503*	Não	Sim
65511	65503	65503	65503	Não	Sim

Tabela 6.10: Resultados do experimento 4

Ao realizar o ataque de subprefixo com a mitigação com anúncio, da mesma forma que no experimento 2, foi observado que no segundo cenário, embora a topologia oferecesse mais opções de caminhos, os ASes *65504*, *65505* e *65506* passaram a acessar a rota verdadeira após a mitigação. Esse comportamento ocorreu pois, ao receber a rota falsa, esses ASes escolheram um caminho que passava pelo *AS-Defender* e, consequentemente, ao receber os pacotes com destino ao atacante o protótipo corrigiu o caminho entregando os dados para a vítima (Vide Tabela 6.11).

AS	CAA	CDA	CDM	Poluído	Mitigado
65501	65502 - 65507 - 65503	interno	interno	Sim	Não
65502	65507 - 65503	65501	65501	Sim	Não
65503	interno	interno	interno	Não	Sim
65504	65507 - 65503	65507 - 65502 - 65501	65507 - 65503	Sim	Sim
65505	65507 - 65503	65507 - 65502 - 65501	65507 - 65503	Sim	Sim
65506	65503	65507 - 65502 - 65501	65507 - 65503*	Sim	Sim
65507	65503	65502 - 65501	65503	Sim	Sim
65508	65509 - 65510 - 65503	65501	65501	Sim	Não
65509	65510 - 65503	65508 - 65501	65508 - 65501	Sim	Não
65510	65503	65509 - 65508 - 65501	65509 - 65508 - 65501	Sim	Não
65511	65503	65503	65503	Não	Sim

Tabela 6.11: Resultados do experimento 5

Os ASes *65508*, *65509* e *65510* utilizaram a rota falsa dado que o caminho até o atacante não tinha o *AS-Defender* como um nó intermediário e, portanto, a rota não pôde ser corrigida.

Para os experimentos 6 e 7 (Tabelas 6.12 e 6.13), o atacante foi trocado com o intuito de observarmos qual seria a quantidade de ASes poluídos ao final do processo de mitigação. Para isso, foram realizados testes em ambos os tipos de mitigação e observada a performance de cada um deles.

AS	CAA	CDA	CDM	Poluído	Mitigado
65501	65502 - 65507 - 65503	65508	65508	Sim	Não
65502	65507 - 65503	65501 - 65508	65507 - 65503	Sim	Sim
65503	interno	interno	interno	Não	Sim
65504	65507 - 65503	65509 - 65508	65507 - 65503	Sim	Sim
65505	65507 - 65503	65510 - 65509 - 65508	65507 - 65503	Sim	Sim
65506	65503	65507 - 65504 - 65509 - 65508	65507 - 65503*	Sim	Sim
65507	65503	65504 - 65509 - 65508	65503	Sim	Sim
65508	65509 - 65510 - 65503	interno	interno	Sim	Não
65509	65510 - 65503	65508	65508	Sim	Não
65510	65503	65509 - 65508	65505 - 65507 - 65503*	Sim	Sim
65511	65503	65503	65503	Não	Sim

Tabela 6.12: Resultados do experimento 6

AS	CAA	CDA	CDM	Poluído	Mitigado
65501	65502 - 65507 - 65503	65508	65508	Sim	Não
65502	65507 - 65503	65501 - 65508	65501 - 65508	Sim	Não
65503	interno	interno	interno	Não	Sim
65504	65507 - 65503	65509 - 65508	65509 - 65508	Sim	Não
65505	65507 - 65503	65510 - 65509 - 65508	65510 - 65509 - 65508	Sim	Não
65506	65503	65507 - 65502 - 65501 - 65508	65507 - 65503*	Sim	Sim
65507	65503	65502 - 65501 - 65508	65503	Sim	Sim
65508	65509 - 65510 - 65503	interno	interno	Sim	Não
65509	65510 - 65503	65508	65508	Sim	Não
65510	65503	65509 - 65508	65509 - 65508	Sim	Não
65511	65503	65503	65503	Não	Sim

Tabela 6.13: Resultados do experimento 7

Na mitigação com anúncio (Tabela 6.12), apesar de o ataque de subprefixo poluir praticamente toda a topologia, ao final da mitigação quase todos os ASes foram recuperados para a rota correta. Isso se deveu em função da posição privilegiada do *AS-Defender* em relação ao atacante, fazendo com que o seu anúncio fosse o caminho preferido pela grande maioria dos ASes. Os ASes *65508* e *65501* não foram mitigados em função da sua proximidade com o *AS65508*, tornando inviável a correção da sua rota através de um anúncio igual ao do atacante.

O mesmo resultado não foi observado na mitigação sem anúncio (Vide Tabela 6.13), posto que, para que a correção de rota para a vítima tenha sucesso, é necessário

que o caminho para o atacante tenha o protótipo como um dos nós intermediários. A posição do atacante gerou um caminho alternativo em direção à rota falsa fazendo com que, excetuando-se o próprio *AS-Defender*, apenas o *AS65506* tivesse a sua rota corrigida durante o processo de mitigação (e ainda assim mais longa em relação a antes do ataque).

A ferramenta também foi testada em uma situação onde um atacante sequestra o prefixo de dois ASes simultaneamente. Embora essa situação não seja algo tão comum, ela foi testada para validarmos como o mecanismo se comportaria ao enfrentar um ataque a dois clientes e precisasse redirecionar cada fluxo para a respectiva vítima. Para atender a esse objetivo, o *AS65509* foi escolhido como o atacante devido à sua posição estratégica entre as duas vítimas, além de ser um dos ASes com o maior número de conexões na topologia, um fator potencializador do ataque.

AS	Destino	CAA	CDA	CDM	Poluído	Mitigado
65501	65503	65502 - 65507 - 65503	65508 - 65509	65502 - 65507 - 65503	Sim	Sim
	65504	65502 - 65507 - 65504	65508 - 65509	65502 - 65507 - 65504	Sim	Sim
65502	65503	65507 - 65503	65507 - 65504 - 65509	65507 - 65503	Sim	Sim
	65504	65507 - 65504	65501 - 65508 - 65509	65507 - 65504	Sim	Sim
65503	65503	interno	interno	interno	Não	Sim
	65504	65507 - 65504	65510 - 65509	65507 - 65504	Sim	Sim
65504	65503	65507 - 65503	65509	65507 - 65503	Sim	Sim
	65504	interno	interno	interno	Não	Sim
65505	65503	65507 - 65503	65510 - 65509	65507 - 65503	Sim	Sim
	65504	65507 - 65504	65510 - 65509	65507 - 65504	Sim	Sim
65506	65503	65503	65507 - 65504 - 65509	65507 - 65503*	Sim	Sim
	65504	65507 - 65504	65503 - 65510 - 65509	65507 - 65504	Sim	Sim
65507	65503	65503	65504 65509	65503	Sim	Sim
	65504	65504	65505 - 65510 - 65509	65504	Sim	Sim
65508	65503	65509 - 65510 - 65503	65509	65509	Sim	Não
	65504	65509 - 65504	65509	65509	Sim	Não
65509	65503	65510 - 65503	interno	interno	Sim	Não
	65504	65504	interno	interno	Sim	Não
65510	65503	65503	65509	65509	Sim	Não
	65504	65509 - 65504	65509	65509	Sim	Não
65511	65503	65503	65503	65503	Não	Sim
	65504	65503 - 65507 - 65504	65503 - 65510 - 65509	65503 - 65507 - 65504	Sim	Sim

Tabela 6.14: Resultados do experimento 8

Novamente a mitigação com anúncio, representada na Tabela 6.14, se mostrou bastante satisfatória ao atrair o tráfego de muitos ASes para o *AS-Defender*, que os encaminhou para o respectivo cliente. Da mesma forma que em outros testes da mitigação com anúncio, apenas o atacante e os dois ASes próximos a ele não voltaram a optar pela rota verdadeira para a vítima. O *AS65511* pela primeira vez

seguiu pela rota falsa, dado que, enquanto os testes eram realizados apenas com o *AS 65503* como vítima, nenhuma das vezes a poluição ocorreu, em função daquele ser vizinho e possuir apenas uma única conexão justamente com este.

AS	Destino	CAA	CDA	CDM	Poluído	Mitigado
65501	65503	65502 - 65507 - 65503	65508 - 65509	65508 - 65509	Sim	Não
	65504	65502 - 65507 - 65504	65508 - 65509	65508 - 65509	Sim	Não
65502	65503	65507 - 65503	65507 - 65504 - 65509	65507 - 65503	Sim	Sim
	65504	65507 - 65504	65501 - 65508 - 65509	65501 - 65508 - 65509	Sim	Não
65503	65503	interno	interno	interno	Não	Sim
	65504	65507 - 65504	65510 - 65509	65510 - 65509	Sim	Não
65504	65503	65507 - 65503	65509	65509	Sim	Não
	65504	interno	interno	interno	Não	Sim
65505	65503	65507 - 65503	65510 - 65509	65510 - 65509	Sim	Não
	65504	65507 - 65504	65510 - 65509	65510 - 65509	Sim	Não
65506	65503	65503	65507 - 65504 - 65509	65507 - 65503*	Sim	Sim
	65504	65507 - 65504	65503 - 65510 - 65509	65503 - 65510 - 65509	Sim	Não
65507	65503	65503	65504 65509	65503	Sim	Sim
	65504	65504	65505 - 65510 - 65509	65504	Sim	Sim
65508	65503	65509 - 65510 - 65503	65509	65509	Sim	Não
	65504	65509 - 65504	65509	65509	Sim	Não
65509	65503	65510 - 65503	interno	interno	Sim	Não
	65504	65504	interno	interno	Sim	Não
65510	65503	65503	65509	65509	Sim	Não
	65504	65509 - 65504	65509	65509	Sim	Não
65511	65503	65503	65503	65503	Não	Sim
	65504	65503 - 65507 - 65504	65503 - 65510 - 65509	65503 - 65510 - 65509	Sim	Não

Tabela 6.15: Resultados do experimento 9

A necessidade do fluxo de dados passar pelo *AS-Defender* não proporcionou uma alta taxa de correção do caminho dos ASes através da mitigação sem anúncio. Quase todos os ASes permaneceram utilizando a rota falsa, mostrando que apenas o ajuste dos fluxos passantes pelo protótipo não é o suficiente para garantir que os ASes em geral utilizem o caminho correto para a vítima.

Também foi experimentada a situação onde o atacante (*AS65510*) era um vizinho da vítima, afim de potencializarmos ainda mais o ataque e identificarmos uma possível relevância do posicionamento do atacante em relação à vítima para o sucesso de um ataque. Os resultados desse experimento podem ser visualizados na Tabela 6.16.

AS	CAA	CDA	CDM	Poluído	Mitigado
65501	65502 - 65507 - 65503	65508 - 65509 - 65510	65502 - 65507 - 65503	Sim	Sim
65502	65507 - 65503	65507 - 65505 - 65510	65507 - 65503	Sim	Sim
65503	interno	interno	interno	Não	Sim
65504	65507 - 65503	65509 - 65510	65507 - 65503	Sim	Sim
65505	65507 - 65503	65510	65507 - 65503	Sim	Sim
65506	65503	65507 - 65505 - 65510	65507 - 65503*	Sim	Sim
65507	65503	65505 - 65510	65503	Sim	Sim
65508	65509 - 65510 - 65503	65509 - 65510	65509 - 65510	Sim	Não
65509	65510 - 65503	65510	65510	Sim	Não
65510	65503	interno	interno	Sim	Não
65511	65503	65503	65503	Sim	Sim

Tabela 6.16: Resultados do experimento 10

Para o experimento 11 (Vide Tabela 6.17), foi avaliada a situação onde um dos clientes do *AS-Defender* fazia o papel do atacante, anunciando o prefixo de outro cliente. Para esse teste, o *AS65503* foi escolhido por possuir o maior número de sessões BGP, possibilitando o anúncio falso ser propagado através de diversos caminhos na topologia.

AS	CAA	CDA	CDM	Poluído	Mitigado
65501	65502 - 65507 - 65504	65502 - 65507 - 65503	65502 - 65507 - 65504	Sim	Sim
65502	65507 - 65504	65507 - 65503	65507 - 65504	Sim	Sim
65503	65507 - 65504	interno	interno	Sim	Não
65504	interno	interno	interno	Não	Sim
65505	65507 - 65504	65507 - 65503	65507 - 65504	Sim	Sim
65506	65507 - 65504	65503	65507 - 65504	Sim	Sim
65507	65504	65503	65504	Sim	Sim
65508	65509 - 65504	65509 - 65510 - 65503	65501 - 65502 - 65507 - 65504*	Sim	Sim
65509	65504	65510 - 65503	65510 - 65503	Sim	Não
65510	65509 - 65504	65503	65503	Sim	Não
65511	65503 - 65507 - 65504	65503	65503	Sim	Não

Tabela 6.17: Resultados do experimento 11

Nesse experimento, pôde ser observado, que o fato de ter sido configurado um filtro de prefixos na vítima, impedindo esta de aceitar anúncios com destino aos prefixos de sua propriedade, pode ter prejudicado a mitigação do *AS65509*. Ao verificarmos o segundo cenário, podemos identificar que o fato do *AS65504* não aceitar tais anúncios, inabilita justamente a rota verdadeira, fazendo com que, através do roteamento por um prefixo mais específico, o caminho preferido seja em direção ao atacante. Todavia, conforme mencionado na Seção 6.2, não seria possível a retirada

do filtro, sob pena da entrega dos dados ao cliente não funcionar adequadamente. Em geral, o comportamento dos ASes ocorreu de uma forma parecida com os demais experimentos onde a mitigação com anúncio foi testada, restando apenas o atacante e os seus ASes mais próximos optando pela rota falsa.

Nos experimentos onde a mitigação sem anúncio foi utilizada, constatou-se que a quantidade de ASes que optaram pela rota verdadeira foi bastante reduzida. Esses resultados forneceram um indício de que apenas os fatos de o *AS-Defender* possuir diversas conexões, estar estrategicamente posicionado e desviar o fluxo de dados em relação a respectiva vítima , não são suficientes para realizar uma mitigação eficiente. Diante disso, foi investigada a possibilidade do uso de recursos adicionais na tentativa de aumentar a eficiência do mecanismo, pelo menos para os clientes do *AS-Defender*. Para alcançar esse objetivo, o experimento 7 serviu como base, dado que os seus resultados (Vide Tabela 6.13) foram os menos eficientes onde a mitigação sem anúncio foi utilizada.

Para tentar induzir uma melhora no processo, foi utilizado um filtro de prefixos BGP em cada um dos clientes (exceto o *AS65502*), que permite o recebimento de anúncios dos subprefixos da vítima somente a partir do *AS-Defender*. Essa mudança obrigou todos os ASes clientes a passar pelo *AS-Defender* durante o ataque e também após a mitigação, conforme mostrado na Tabela 6.18.

AS	**CAA**	**CDA**	**CDM**	**Poluído**	**Mitigado**
65501	65502 - 65507 - 65503	65508	65508	Sim	Não
65502	65507 - 65503	65501 - 65508	65501 - 65508	Sim	Não
65503	interno	interno	interno	Não	Sim
65504	65507 - 65503	65507 - 65502 - 65501 - 65508	65507 - 65503	Sim	Sim
65505	65507 - 65503	65507 - 65502 - 65501 - 65508	65507 - 65503	Sim	Sim
65506	65503	65507 - 65502 - 65501 - 65508	65507 - 65503	Sim	Sim
65507	65503	65502 - 65501 - 65508	65503	Sim	Sim
65508	65509 - 65510 - 65503	interno	interno	Sim	Não
65509	65510 - 65503	65508	65508	Sim	Não
65510	65503	65509 - 65508	65509 - 65508	Sim	Não
65511	65503	65503	65503	Não	Sim

Tabela 6.18: Resultados do experimento 12

Os resultados mostraram que todos os clientes, com exceção do AS *65502*, passaram a optar pela rota verdadeira após a mitigação. Não foi possível utilizar o filtro no *AS65502* em razão da sua conexão iBGP com o coletor de rotas pois, quando o filtro foi aplicado, os anúncio não chegavam ao coletor e, consequentemente, os

ataques não eram identificados pelo protótipo. Em razão da impossibilidade do uso ter sido por um requisito estritamente técnico relacionado ambiente experimental, para efeitos de um ambiente BGP real, também seria possível influenciar o *AS65502* a utilizar a rota verdadeira.

6.3.3 Resumo

Os resultados mostraram que os mecanismos realizados pelo *AS-Defender* para combater o sequestro de IP funcionaram em ambos os cenários abordados. Ao mesmo tempo, a mitigação sem anúncio se mostrou com uma baixa eficácia quando a topologia foi expandida no segundo cenário. Através de um recurso de filtragem de anúncios foi possível melhorar os seus resultados forçando os ASes clientes a seguirem um caminho que passava pelo *AS-Defender* para que o mesmo pudesse direcionar os dados à vítima. Ao experimentar a mitigação sem anúncio, não era uma intenção do presente trabalho que este método pudesse superar os resultados da versão com anúncio, mas sim, entender se pelo menos os ASes mais próximos poderiam ser mitigados de modo que, caso esse mecanismo fosse adotado por diversos ASes, seria possível criar uma cadeia de mitigação até o destino.

Os ataques de prefixo, abordados nos experimentos 1 e 4, embora não tenham sido largamente explorados, confirmaram alguns estudos [7, 9] que sugerem uma reduzida taxa de poluição nos ASes, comparado aos ataques de subprefixo. Até mesmo em função desses resultados, o foco dos experimentos se deu nos ataques de subprefixo, com a intenção de colocarmos o protótipo sob as condições mais adversas possíveis.

Em alguns casos, após a mitigação com anúncio, foi identificado que o caminho utilizado por um AS para a vítima continha mais saltos que o caminho originalmente escolhido. Esse comportamento era até certo ponto esperado dado que, o fato do *AS-Defender* buscar atrair o tráfego para si, não tem como objetivo nenhum tipo de garantia de entrega dos dados através de um caminho mais curto.

O filtro de prefixo inserido em cada vítima para o não recebimento dos próprios subprefixos evitou que as rotas anunciadas pelo *AS-Defender* chegassem a alguns locais da topologia por meio de alguns enlaces que poderiam possibilitar uma mitigação mais efetiva.

Sob uma ótica abrangente, os efeitos da detecção e da mitigação proporcionados pelo protótipo mostraram que a arquitetura SDN pode contribuir para o combate

ao sequestro de prefixo no BGP. Corroborando essa análise, a Tabela 6.19 mostra uma visão geral de todos os experimentos realizados, reafirmando os itens previamente discutidos. As colunas “Poluídos”, “Mitigados” e “Rota correta” representam respectivamente a quantidade de ASes que optaram pela rota falsa após o ataque, a quantidade de ASes poluídos que voltaram a preferir a rota para a vítima após a mitigação e o total de ASes que utilizaram a rota correta ao final de todo o ciclo de defesa.

Nº exp	**Atacante**	**Vítima**	**Ataque**	**Mitigação**	**Poluídos**	**Mitigados**	**Rota correta**
1	AS 65501	AS 65503	prefixo	Com anúncio	2	2	7
2	AS 65501	AS 65503	subprefixo	Sem anúncio	6	4	5
3	AS 65501	AS 65503	subprefixo	Com anúncio	6	5	6
4	AS 65501	AS 65503	prefixo	Com anúncio	4	4	11
5	AS 65501	AS 65503	subprefixo	Sem anúncio	9	4	6
6	AS 65508	AS 65503	subprefixo	Com anúncio	9	6	8
7	AS 65508	AS 65503	subprefixo	Sem anúncio	9	2	4
8	AS 65509	AS 65503	subprefixo	Com anúncio	9	6	8
		AS 65504	subprefixo	Com anúncio	10	7	8
9	AS 65509	AS 65503	subprefixo	Sem anúncio	9	3	5
		AS 65504	subprefixo	Sem anúncio	10	1	2
10	AS 65510	AS 65503	subprefixo	Com anúncio	10	7	8
11	AS 65503	AS 65504	subprefixo	Com anúncio	10	6	7
12	AS 65508	AS 65503	subprefixo	Sem anúncio	9	4	6

Tabela 6.19: Visão geral dos resultados dos experimentos

6.4 Limitações

Uma primeira limitação do *AS-Defender* é que, em função do algoritmo de detecção ser uma implementação parcial do utilizado no projeto ARTEMIS, alguns tipos de sequestros ainda não são identificados pelo protótipo, como os ataques do Tipo 2 em diante e o *squatting*. Uma outra limitação é que, embora o número de subprefixos para o anúncio da mitigação seja ilimitado, o protótipo monitora apenas um prefixo por cliente, devido a forma como é feito o armazenamento das informações obtidas a partir dos arquivos de configuração dos clientes.

O protótipo também não possui a funcionalidade de identificar que um ataque foi interrompido e desfazer todas as mudanças executadas previamente na mitigação. Essa funcionalidade foi analisada e inicialmente o objetivo era utilizar as mensagens *WITHDRAW* para identificar o término do ataque. Porém, dado que ao analisarmos a sua estrutura, não foi encontrada uma forma de saber que a retirada do

anúncio foi realmente propagada pelo atacante, essa solução não foi adotada. Em contrapartida, foi percebido que, quando um AS anuncia um prefixo como de sua propriedade, a sua tabela de roteamento BGP contém essa informação e, dado que o *AS-Defender* possui a capacidade de identificar o AS que executou o ataque, através do monitoramento da sua tabela de roteamento, talvez seja possível implementar esse recurso.

Para mitigar um ataque, os subprefixos a serem anunciados são obtidos através do arquivo de configuração dos clientes, porém, para que qualquer cenário de ataque seja apropriadamente mitigado, o ideal é que os prefixos sejam desagregados sem a necessidade de serem previamente configurados. Assim, ao receber um anúncio de ataque, a mitigação automaticamente divide o prefixo em subprefixos imediatamente mais específicos, realizando o que alguns trabalhos chamam de desagregação automática de prefixos [20, 19]. Tal recurso ainda não está presente na implementação do *AS-Defender*.

Para a utilização do *AS-Defender* em um cenário real, alguns pontos sobre como o protótipo foi implementado precisam ser levados em consideração. Um desses pontos é que, em um cenário de BGP tradicional, uma quantidade grande de dados entre os ASes é trocada e, devido à alta granularidade das regras *OpenFlow* que são inseridas no *switch*, as tabelas de fluxos podem alcançar um tamanho enorme, gerando problemas de escalabilidade e performance. Ademais, devido ao curto tempo de expiração das entradas de fluxo inseridas pelo protótipo na fase de operação, ao se utilizar uma malha de *switches* no plano de dados, deve-se ter em mente que isso poderá acarretar um grande número de requisições ao controlador, gerando uma sobrecarga no plano de controle.

A escalabilidade da solução é um aspecto a ser estudado, dado que, conforme alguns trabalhos, o próprio paradigma SDN ainda possui limitações relacionadas ao tamanho das tabelas de fluxo e à quantidade de mensagens *PacketIn* direcionadas ao controlador. Por exemplo, para cenários onde o *AS-Defender* seja utilizado em grandes ISP´s, podem ser necessárias mudanças no código do protótipo afim de melhorar a sua performance, bem como o uso de um controlador adicional para dividir a carga de trabalho demandada pelos *switches*.

Nos experimentos não foi analisada a performance do *AS-Defender* em cenários onde um grande serviço fosse afetado e, consequentemente, uma grande quantidade de dados fosse atraída e redirecionada ao destino. Situações como essa podem não obter um desempenho adequado e seria interessante que fossem objeto de estudo em

outras oportunidades para mostrar a adequação do modelo proposto em episódios de grandes proporções.

Em um ambiente real, a detecção do ataque passaria a ser feita a partir das informações públicas de roteamento BGP fornecidas por alguns projetos na internet. Essa diferença incorreria em uma reprogramação da *thread* de monitoramento e do código de detecção, porque tanto a forma de conexão, as fontes de informação e a estrutura de dados recebida no monitoramento do plano de controle seriam diferentes do atualmente utilizado nos cenários experimentais.

Inferindo o funcionamento do *AS-Defender* em um AS com poucas conexões, podemos acreditar que tal cenário poderia gerar uma limitação nas funcionalidades de mitigação do ataque, dado que, ao configurarmos o filtro de prefixo nos clientes, o anúncio de mitigação já não é encaminhado pelo caminho onde o respectivo cliente está, reduzindo o número de enlaces onde o anúncio é propagado.

Nenhuma das limitações acima descritas foram suficientemente impactantes no que tange à realização da prova de conceito do protótipo, porém precisam ser superadas para a materialização de trabalhos futuros e melhorias no projeto.

6.5 Síntese

Este capítulo apresentou os cenários utilizados na validação do protótipo além de discutir o resultado de cada experimento realizado. Na sequência, os resultados foram comentados e resumidos em um aspecto macroscópico e as limitações do protótipo foram discutidas com o fornecimento de possíveis soluções para algumas delas.

7. Conclusão

Este capítulo resume o presente trabalho, além de abordar os próximos caminhos a serem seguidos para a continuidade da sua evolução, bem como as considerações finais sobre o projeto e uma breve reflexão sobre as contribuições fornecidas por este estudo.

Para a concepção deste trabalho, foi necessária uma extensa exploração da literatura sobre o estado da arte a respeito do *prefix hijacking*. Este estudo identificou a possibilidade de juntar os benefícios inerentes ao SDN com formas existentes de combate ao ataque. A união das redes definidas por *software*, através do controlador *Ryu*, com algumas técnicas de detecção e mitigação existentes no ARTEMIS possibilitou a criação do *AS-Defender*: um sistema autônomo para proteção dos seus vizinhos por meio da detecção e mitigação do ataque de sequestro de prefixo. Em sua implantação, uma *thread* de monitoramento instancia o mecanismo de detecção para a identificação de alguns tipos de ataques e recebe os dados dos anúncios através do módulo coletor de rotas, que faz o papel dos serviços de disponibilização de dados públicos do plano de controle do BGP.

O *AS-Defender* opera as diversas fases de funcionamento concorrentemente, cada uma delas com um papel definido no ciclo de proteção provido pelo protótipo. Todas as atividades de monitoramento e mitigação ocorrem paralelamente ao funcionamento da comunicação entre o protótipo e os seus vizinhos, mediante um processo BGP ativado no controlador, possibilitando o roteamento dos pacotes entre os sistemas autônomos. A proteção dos vizinhos é apoiada por um arquivo de configuração no padrão JSON, onde as informações do prefixo monitorado, os subprefixos a serem anunciados, o IP do cliente, assim como outros dados, alimentam o protótipo e possibilitam a decisão sobre a forma como cada cliente deve ser defendido.

O protótipo realiza dois tipos de mitigação: com anúncio e sem anúncio. En-

quanto a primeira visa atrair a maior quantidade possível de tráfego direcionada aos subprefixos dos clientes, a segunda serve como um *baseline* de comparação, um teste para validação da necessidade de anunciar os subprefixos do cliente, e também para os casos onde um vizinho já possua um mecanismo de defesa próprio.

Os resultados obtidos a partir da validação experimental mostraram que tanto a detecção quanto a mitigação do ataque aos clientes foram executadas com sucesso, assegurando a prova de conceito como funcional, atendendo aos objetivos descritos na Seção 1.2 e proporcionando as seguintes contribuições:

- O fornecimento de um mecanismo de combate ao *prefix hijacking* onde múltiplos ASes podem ser protegidos através da combinação de tecnologias e recursos criados em outro trabalhos.
- A verificação de que uma mitigação apenas com o redirecionamento dos dados passantes por um AS, por si só, não combate totalmente o ataque, sendo necessário mecanismos complementares que forneçam a passagem dos dados pelo AS defensor.
- A contribuição implícita a respeito dos conhecimentos gerados pelos mecanismos criados, por serem úteis na implantação de um AS defensor que possa interagir com outros controladores.
- A criação de um módulo de gerenciamento de um *switch* que, embora precise ser ajustado em alguns casos, possa ser consumido por outras aplicações SDN que precisem realizar operações semelhantes.
- A comprovação de que a união de diversas técnicas de combate ao ataque podem ser eficazes, onde foi demonstrado que o uso de um filtro de prefixo, aliado à mitigação sem anúncio, pode garantir o acesso à vítima pelo menos para os vizinhos do *AS-Defender*.

Dentre as contribuições fornecidas pelo projeto *AS-Defender*, a mitigação possui uma relevância maior, dado que poucos trabalhos ainda exploram esse aspecto. Portanto, esta pesquisa tem como principal colaboração as formas de mitigação do ataque de *prefix hijacking* através do uso das redes definidas por *software*.

7.1 Trabalhos futuros

Durante a implementação do protótipo, foram tomadas decisões que priorizaram a execução dos experimentos, deixando alguns aprimoramentos para desenvolvimentos futuros. Esses desenvolvimentos visam tornar o protótipo mais aderente a uma utilização em um cenário real, além de possibilitar o seu uso nos ambientes BGP mais heterogêneos possíveis. Alguns desses aprimoramentos podem ser implementados em um curto prazo, porém outros demandam mais tempo ou podem necessitar um estudo mais aprofundado sobre como executa-los. A seguinte lista abrange os desenvolvimentos futuros:

- Implementação do código para detecção dos ataques Tipo 2 em diante e *squatting*. Essa implementação tornaria o algoritmo de detecção mais versátil e robusto, atendendo a mais cenários de ataques. Após alguns estudos, a identificação do *squatting* parece ser possível através do uso do mesmo arquivo de configuração de clientes utilizado atualmente pelo protótipo. Quanto aos ataques Tipo 2 em diante, como cada cliente não possui a visão da sequência de ASes que estão conectados após os seus vizinhos, seria necessário implementar o algoritmo utilizado pelo projeto ARTEMIS, que armazena a sequência do *AS Path* nos anúncios e, após um tempo, passa a comparar com os anúncios que são propagados na rede. Uma outra opção seria realizar um estudo para a criação de um algoritmo próprio, o que poderia demandar bastante tempo. Esse recurso não foi implementado no presente trabalho em função da documentação disponibilizada em artigos não terem sido suficiente para o entendimento do seu funcionamento.

- Implementação do recurso de desagregação automática de prefixos. Este recurso, à princípio, eliminaria a necessidade do uso do atributo *SUBPREFIXES* no arquivo de configuração do cliente pois, ao verificar um anúncio falso, o prefixo anunciado seria automaticamente dividido em sub-redes para a mitigação, sem a necessidade de consulta ao arquivo.

- Implementação de um método para o *rollback* da mitigação após a detecção do término de um ataque. Para retornar da fase de mitigação, conforme dito na Seção 6.4, a tabela de roteamento do atacante poderia ser monitorada afim de, ao identificar que o prefixo utilizado no ataque não corresponde mais a um prefixo de sua propriedade, realizar o retorno das ações de mitigação executadas.

- Expandir a quantidade de prefixos protegidos por cliente. Esse aprimoramento habilitaria o monitoramento de mais de um prefixo alvo para cada AS protegido. Embora isso não impacte em nenhum experimento nem em um futuro uso do protótipo em um cenário real, está considerado para uma futura realização.

- Realização de novos experimentos em um cenário real. Os testes em um cenário real visam expor o protótipo a comunicações com ASes existentes na Internet, avaliando se os resultados obtidos no ambiente controlado são ratificados ou discrepantes. Adicionalmente, será necessário avaliar se será necessário trocar a ferramenta Knet, utilizada na construção da topologia, ou se esta possui a possibilidade de se integrar a ambientes na Internet. Em virtude do ambiente ser baseado em contêineres, acreditamos que essa integração seja possível através dos recursos do *docker*.

- Avaliar o uso de dois protótipos simultaneamente. Essa avaliação serviria para identificarmos possíveis inconsistências geradas no roteamento ao configurarmos mais de um *AS-Defender* atuando no mesmo ambiente. Testar esse cenário demanda mudanças na topologia construída, além executar duas instâncias do Ryu, cada uma se comunicando com o respectivo *switch*.

7.2 Considerações finais

O *AS-Defender* foi fruto de um estudo multidisciplinar e possui relação com alguns trabalhos envolvendo as redes definidas por *software* e o BGP. Por ser a tecnologia que "cola" toda a internet, o BGP possui um papel primordial na comunicação entre os sistemas autônomos, tornando desafiadora a busca por soluções a respeito do seu funcionamento.

Diante da complexidade do assunto, os estudos acerca do ataque têm atuado em diversas frentes de solução para o problema. Enquanto alguns focam na sua detecção e mitigação, outros têm proposto mudanças no próprio protocolo, dificultando uma implantação em escala global.

Ao mostrar o potencial do paradigma SDN no combate ao ataque *prefix hijacking*, através de um AS protetor de seus vizinhos, julgamos termos cumprido as contribuições projetadas e esperamos auxiliar na abertura de um canal com outras possibilidades, através da combinação do uso de mecanismos já existentes com pa-

radigmas ou tecnologias que nos possibilitem abordar o problema sob uma ótica diferente.

Referências Bibliográficas

[1] K. James and R. Keith, "Redes de computadores e a internet: Uma abordagem top-down," 2005.

[2] G. Huston, M. Rossi, and G. Armitage, "Securing bgp—a literature survey," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 2, pp. 199–222, 2011.

[3] M. Pokornỳ and J. Zima, "Data path definition in software defined networks,"

[4] O. N. Foundation, "OpenFlow Switch Specification Version 1.3.0 (Wire Protocol 0x04)." https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf, 2012. [Online; accessed 28-Jul-2018].

[5] R. Soares and S. C. d. L. Silva, "Havox: Um serviço para orquestraç ao de tráfego em alto nıvel em redes openflow,"

[6] P. Goransson, C. Black, and T. Culver, *Software defined networks: a comprehensive approach*. Morgan Kaufmann, 2016.

[7] Z. Zhang, Y. Zhang, Y. C. Hu, and Z. M. Mao, "Practical defenses against bgp prefix hijacking," in *Proceedings of the 2007 ACM CoNEXT conference*, p. 3, ACM, 2007.

[8] P. Traina, "Bgp-4 protocol analysis," 1995.

[9] C. Horn, "Understanding ip prefix hijacking and its detection," in *Seminar internet routing, intelligent networks (INET)*, 2009.

[10] P. Sermpezis, V. Kotronis, A. Dainotti, and X. Dimitropoulos, "A survey among network operators on bgp prefix hijacking," *arXiv preprint arXiv:1801.02918*, 2018.

[11] Z. Zhang, Y. Zhang, Y. C. Hu, Z. M. Mao, and R. Bush, "Ispy: detecting ip prefix hijacking on my own," in *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 327–338, ACM, 2008.

[12] X. Hu and Z. M. Mao, "Accurate real-time identification of ip prefix hijacking," in *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pp. 3–17, IEEE, 2007.

[13] W. Aiello, J. Ioannidis, and P. McDaniel, "Origin authentication in interdomain routing," in *Proceedings of the 10th ACM conference on Computer and communications security*, pp. 165–178, ACM, 2003.

[14] Y.-C. Hu, A. Perrig, and M. Sirbu, "Spv: Secure path vector routing for securing bgp," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 179–192, 2004.

[15] K. Butler, P. McDaniel, and W. Aiello, "Optimizing bgp security by exploiting path stability," in *Proceedings of the 13th ACM conference on Computer and communications security*, pp. 298–310, ACM, 2006.

[16] K. Benton, L. J. Camp, and M. Swany, "Bongo: A bgp speaker built for defending against bad routes," in *Military Communications Conference, MILCOM 2016-2016 IEEE*, pp. 735–739, IEEE, 2016.

[17] K. Balu, M. L. Pardal, and M. Correia, "Darshana: Detecting route hijacking for communication confidentiality," in *Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on*, pp. 52–59, IEEE, 2016.

[18] C. Zheng, L. Ji, D. Pei, J. Wang, and P. Francis, "A light-weight distributed scheme for detecting ip prefix hijacks in real-time," in *ACM SIGCOMM Computer Communication Review*, vol. 37, pp. 277–288, ACM, 2007.

[19] P. Sermpezis, G. Chaviaras, P. Gigis, and X. Dimitropoulos, "Monitor, detect, mitigate: Combating bgp prefix hijacking in real-time with artemis," *arXiv preprint arXiv:1609.05702*, 2016.

[20] P. Sermpezis, V. Kotronis, P. Gigis, X. Dimitropoulos, D. Cicalese, A. King, and A. Dainotti, "Artemis: Neutralizing bgp hijacking within a minute," *arXiv preprint arXiv:1801.01085*, 2018.

[21] RouteViews, "Route Views." http://www.routeviews.org/routeviews/. [Online; accessed 29-April-2018].

[22] RIPENCC, "RIPE NCC." https://www.ripe.net/. [Online; accessed 29-April-2018].

[23] H. Alshamrani and B. Ghita, "Ip prefix hijack detection using bgp attack signatures and connectivity tracking," in *Software Networking (ICSN), 2016 International Conference on*, pp. 1–7, IEEE, 2016.

[24] H. Alshamrani and B. Ghita, "Ip prefix hijack detection using bgp connectivity monitoring," in *High Performance Switching and Routing (HPSR), 2016 IEEE 17th International Conference on*, pp. 35–41, IEEE, 2016.

[25] S. Goldberg, "Why is it taking so long to secure internet routing?," *Queue*, vol. 12, no. 8, p. 20, 2014.

[26] N. Telegraph and T. Corporation, "Ryu, build sdn agilely." https://osrg.github.io/ryu/, 2018. [Online; accessed 11-Ago-2018].

[27] J. Moy, O. Version, N. W. Group, *et al.*, "Rfc 2328, ascend communications," 1998.

[28] G. Malkin, "Rip version 2," tech. rep., 1998.

[29] Noction, "Understanding the AS path and AS path prepending." https://www.noction.com/blog/as-path-and-as-path-prepending, 2015. [Online; accessed 14-July-2018].

[30] J. Schlamp, M. Wählisch, T. C. Schmidt, G. Carle, and E. W. Biersack, "Cair: Using formal languages to study routing, leaking, and interception in bgp," *arXiv preprint arXiv:1605.00618*, 2016.

[31] M. M. J. M. O. Z. Ondrej Filip, Pavel Machek, "BIRD User´s Guide (version2.0.2)." https://bird.network.cz/?index, 2018. [Online; accessed 07-September-2018].

[32] E. C. Rosen, "Exterior gateway protocol (egp)," tech. rep., 1982.

[33] Juniper, "router-id." https://www.juniper.net/documentation/en_US/junos/topics/reference/statement/router-id-edit-routing-options.html, 2017. [Online; accessed 10-Jul-2018].

[34] T. Qiu, L. Ji, D. Pei, J. Wang, and J. Xu, "Towerdefense: Deployment strategies for battling against ip prefix hijacking," in *Network Protocols (ICNP), 2010 18th IEEE International Conference on*, pp. 134–143, IEEE, 2010.

[35] P. Wani and S. Mali, "Securing routing protocol bgp," 2016.

[36] C. McArthur and M. Guirguis, "Stealthy ip prefix hijacking: don't bite off more than you can chew," in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pp. 1–6, IEEE, 2009.

[37] M. S. Siddiqui, D. Montero, R. Serral-Gracià, X. Masip-Bruin, and M. Yannuzzi, "A survey on the recent efforts of the internet standardization body for securing inter-domain routing," *Computer Networks*, vol. 80, pp. 1–26, 2015.

[38] Nanog, "Another day, another illicit SQUAT." http://seclists.org/nanog/2016/Oct/578, 2016. [Online; accessed 23-Jun-2018].

[39] M. Lad, D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang, "Phas: A prefix hijack alert system.," in *USENIX Security symposium*, vol. 1, p. 3, 2006.

[40] BGPmon, "BGPmon: Using Real-Time Data in Research and Operations." https://www.bgpmon.io/documentation/BGPmon-deployment.pdf. [Online; accessed 29-Jun-2018].

[41] Y.-J. Chi, R. Oliveira, and L. Zhang, "Cyclops: the as-level connectivity observatory," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 5, pp. 5–16, 2008.

[42] J. Qiu, L. Gao, S. Ranjan, and A. Nucci, "Detecting bogus bgp route information: Going beyond prefix hijacking," in *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*, pp. 381–390, IEEE, 2007.

[43] P. Boothe, J. Hiebert, and R. Bush, "How prevalent is pre x hijacking on the internet," *Adresse: http://141.223*, vol. 82, 2005.

[44] X. Shi, Y. Xiang, Z. Wang, X. Yin, and J. Wu, "Detecting prefix hijackings in the internet with argus," in *Proceedings of the 2012 Internet Measurement Conference*, pp. 15–28, ACM, 2012.

[45] H. Yan, R. Oliveira, K. Burnett, D. Matthews, L. Zhang, and D. Massey, "Bgpmon: A real-time, scalable, extensible monitoring system," in *Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*, pp. 212–223, IEEE, 2009.

[46] D. Shytyi and T. Quang-Huy, "Bgp hijack detection via changepoint detection,"

[47] M. Tahara, N. Tateishi, T. Oimatsu, and S. Majima, "A method to detect prefix hijacking by using ping tests," in *Asia-Pacific Network Operations and Management Symposium*, pp. 390–398, Springer, 2008.

[48] A. Mitseva, A. Panchenko, and T. Engel, "The state of affairs in bgp security: A survey of attacks and defenses," *Computer Communications*, 2018.

[49] H. Farhady, H. Lee, and A. Nakao, "Software-defined networking: A survey," *Computer Networks*, vol. 81, pp. 79–95, 2015.

[50] O. N. Foundation, "Software-Defined Networking (SDN) Definition." https://www.opennetworking.org/sdn-definition/, 2017. [Online; accessed 30-Jul-2018].

[51] M. McNickle, "Five SDN protocols other than OpenFlow." https://searchsdn.techtarget.com/news/2240227714/Five-SDN-protocols-other-than-OpenFlow, 2014. [Online; accessed 29-Jul-2018].

[52] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[53] F. M. Ramos, D. Kreutz, and P. Verissimo, "Software-defined networks: On the road to the softwarization of networking," *Cutter IT journal*, 2015.

[54] W. Braun and M. Menth, "Software-defined networking using openflow: Protocols, applications and architectural design choices," *Future Internet*, vol. 6, no. 2, pp. 302–336, 2014.

[55] B. Agborubere and E. Sanchez-Velazquez, "Openflow communications and tls security in software-defined networks," in *Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2017 IEEE International Conference on*, pp. 560–566, IEEE, 2017.

[56] S. Azodolmolky, *Software defined networking with OpenFlow*. Packt Publishing Ltd, 2013.

[57] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[58] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[59] N. Telegraph and T. Corporation, "ryu.app.ofctl_rest." http://ryu.readthedocs.io/en/latest/app/ofctl_rest.html, 2014. [Online; accessed 10-Ago-2018].

[60] O. Salman, I. H. Elhajj, A. I. Kayssi, and A. Chehab, "Sdn controllers: A comparative study," *2016 18th Mediterranean Electrotechnical Conference (MELECON)*, pp. 1–6, 2016.

[61] J. Tavares, H. S. Mamede, P. Amaral, and P. Pinto, "Software-defined controllers: Where are we?," in *Information Systems and Technologies (CISTI), 2017 12th Iberian Conference on*, pp. 1–6, IEEE, 2017.

[62] S. Rowshanrad, S. Namvarasl, V. Abdi, M. Hajizadeh, and M. Keshtgary, "A survey on sdn, the future of networking," *Journal of Advanced Computer Science & Technology*, vol. 3, no. 2, pp. 232–248, 2014.

[63] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of sdn/openflow controllers," in *Proceedings of the 9th central & eastern european software engineering conference in russia*, p. 1, ACM, 2013.

[64] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 13–18, ACM, 2013.

[65] A. B. S. Networks, "Floodlight." http://www.projectfloodlight.org/floodlight/, 2018. [Online; accessed 11-Ago-2018].

[66] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 1–6, ACM, 2014.

[67] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *2014 IEEE 15th International Symposium on*, pp. 1–6, IEEE, 2014.

[68] Trema, "Trema - full-stack openflow framework in ruby and c." https://trema.github.io/trema/, 2018. [Online; accessed 11-Ago-2018].

[69] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[70] O. MUL, "Open mul." http://www.openmul.org/openmul-controller.html, 2018. [Online; accessed 11-Ago-2018].

[71] M. P. Fernandez, "Comparing openflow controller paradigms scalability: Reactive and proactive," in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pp. 1009–1016, IEEE, 2013.

[72] I. Fette and A. Melnikov, "The websocket protocol," tech. rep., 2011.

[73] J. Schlamp, R. Holz, Q. Jacquemart, G. Carle, and E. W. Biersack, "Heap: reliable assessment of bgp hijacking attacks," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 6, pp. 1849–1861, 2016.

[74] B. Schlinker, K. Zarifis, I. Cunha, N. Feamster, and E. Katz-Bassett, "Peering: An as for us," in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, p. 18, ACM, 2014.

[75] CAIDA, "Archipelago." http://www.caida.org/projects/ark/, 2007. [Online; accessed 31-August-2018].

[76] Noction, "Looking Glasses." https://www.noction.com/blog/bgp-looking-glass-servers, 2017. [Online; accessed 31-August-2018].

[77] P. Documentation, "An introduction to the ipaddress module." https://cpython-test-docs.readthedocs.io/en/latest/howto/ipaddress.html, 2016. [Online; accessed 07-September-2018].

[78] PyYAML, "PyYAML." https://pyyaml.org/wiki/PyYAML, 2017-2018. [Online; accessed 07-September-2018].

[79] P. Documentation, "sys — System-specific parameters and functions." https://docs.python.org/2/library/sys.html, 2018. [Online; accessed 07-September-2018].

[80] P. Documentation, "threading — Higher-level threading interface." https://docs.python.org/2/library/threading.html, 2018. [Online; accessed 07-September-2018].

[81] M. Grinberg, "python-socketio." https://github.com/miguelgrinberg/python-socketio, 2018. [Online; accessed 07-September-2018].

[82] R. Labs, "Pub/Sub." https://redis.io/topics/pubsub, 2018. [Online; accessed 07-September-2018].

[83] D. Moss, "netaddr 0.7.19 documentation." https://netaddr.readthedocs.io/en/latest/, 2008. [Online; accessed 07-September-2018].

[84] N. Telegraph and T. Corporation, "Welcome to RYU the Network Operating System(NOS)." https://ryu.readthedocs.io/en/latest/index.html, 2011-2014. [Online; accessed 07-September-2018].

[85] Exa-Networks, "ExaBGP." https://github.com/Exa-Networks/exabgp, 2018. [Online; accessed 07-September-2018].

[86] D. Mavrommatis, "ExaBGP-Monitor." https://github.com/slowr/ExaBGP-Monitor, 2018. [Online; accessed 07-September-2018].

[87] K. Solutions, "Knet." https://github.com/knetsolutions/KNet, 2018. [Online; accessed 07-September-2018].

[88] L. Colitti, G. Di Battista, F. Mariani, M. Patrignani, and M. Pizzonia, "Visualizing interdomain routing with bgplay.," *J. Graph Algorithms Appl.*, vol. 9, no. 1, pp. 117–148, 2005.

[89] Y. Rekhter, S. Hares, and T. Li, "A Border Gateway Protocol 4 (BGP-4)." RFC 4271, Jan. 2006.

[90] E. Chen, "Method and apparatus for dynamic exchange of capabilities between adjacent/neighboring networks nodes," Apr. 22 2003. US Patent 6,553,423.

[91] E. Chen, "Route refresh capability for bgp-4," tech. rep., 2000.

[92] Cisco, "BGP Best Path Selection Algorithm." https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html, 2016. [Online; accessed 09-Jun-2018].

[93] D. Savage, J. Ng, S. Moore, D. Slice, P. Paluch, and R. White, "Cisco's enhanced interior gateway routing protocol (eigrp)," tech. rep., 2016.

[94] L. V. Defcon 16, "Stealing The Internet." https://www.defcon.org/images/defcon-16/dc16-presentations/defcon-16-pilosov-kapela.pdf, 2008. [Online; accessed 25-Jun-2018].

[95] J. Yang, C. Hu, P. Zheng, R. Wang, P. Zhang, and X. Guan, "Rethinking the design of openflow switch counters," in *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 589–590, ACM, 2016.

[96] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takács, and P. Sköldström, "Scalable fault management for openflow," in *Communications (ICC), 2012 IEEE international conference on*, pp. 6606–6610, IEEE, 2012.

[97] M. Reitblatt, M. Canini, A. Guha, and N. Foster, "Fattire: Declarative fault tolerance for software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 109–114, ACM, 2013.

[98] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann, "Ofrewind: enabling record and replay troubleshooting for networks," in *USENIX Annual Technical Conference*, pp. 327–340, USENIX Association, 2011.

[99] O. N. Foundation, "OpenFlow Switch Specification Version 1.0.0 (Wire Protocol 0x01)." https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf, 2009. [Online; accessed 08-Ago-2018].

[100] C. Orsini, A. King, D. Giordano, V. Giotsas, and A. Dainotti, "Bgpstream: a software framework for live and historical bgp data analysis," in *Proceedings of the 2016 Internet Measurement Conference*, pp. 429–444, ACM, 2016.

[101] R. NCC, "RIPEstat." https://stat.ripe.net/. [Online; accessed 29-August-2018].