BotFetcher: Um Método Híbrido de Detecção de *Botnets*

Bruno Martins Rahal[1], **Alissar Ali Moussa**[1], **Michele Nogueira**[1]

[1]Centro de Ciência de Segurança Computacional (CCSC)
Universidade Federal do Paraná (UFPR)

{bmrahal,aamoussa,michele} at inf.ufpr.br

Resumo. *Este trabalho apresenta BotFetcher, um método híbrido para detecção de botnets que considera técnicas de agrupamento e processamento de sinais em grafos. Botnets representam uma ameaça a redes de computadores, pois podem interromper seus serviços através da coordenação entre uma quantidade massiva de dispositivos infectados (bots), com prejuízos globais estimados em US$ 5,8 bilhões em 2019. BotFetcher contribui para a detecção em escala de botnets, com grande volume de tráfego, principalmente aquelas formadas por dispositivos da IoT infectados, como as botnets Mirai, Hydra e LuaBot. A detecção dessas botnets é desafiadora devido à quantidade massiva de dados e às limitações de processamento e memória. Assim, BotFetcher agrupa os dispositivos de rede com base em características extraídas do tráfego e realiza a detecção por grupo a partir de indícios de causalidade entre os dispositivos. A avaliação do método utilizou a base de dados CTU-13 da Universidade da República Tcheca. BotFetcher detectou o bot no cenário 5 da base CTU-13, com 55 falsos-positivos e nenhum falso-negativo, entre os 39738 nós avaliados.*

1. Introdução

O uso da Internet tem crescido e atinge cada vez mais pessoas, principalmente, com o advento e a popularização da Internet das Coisas (IoT). Estatísticas mostram que, em junho de 2019, cerca de 60% da população mundial (aproximadamente 4,5 bilhões de pessoas) estava conectada a esta rede [Miniwatts Marketing Group 2019]. Consequentemente, há um número crescente de dispositivos (nós) conectados e um aumento no volume tráfego da rede, com a previsão de crescimento de 27% ao ano até 2022, triplicando efetivamente o tráfego em 5 anos [Cisco 2018]. Além disso, a Internet abriga dados sensíveis pertencentes a usuários (*e.g.*, dados bancários, conversas confidenciais e outras), e a grandes corporações. Dessa forma, mecanismos de proteção contra ataques são necessários para evitar danos à integridade dos dados, à disponibilidade de serviços e sistemas, e também impedir roubos de informações sensíveis.

Os atacantes dispõem de diversas ferramentas e artifícios para gerar e executar os mais variados tipos de ataques na Internet, dentre as quais estão as *botnets*. Estas são redes de dispositivos infectados (*bots*), em que um ou mais dispositivos mestres (*botmasters*) os comandam. Estas redes se beneficiam do funcionamento adequado da rede de computadores para realizar, por exemplo, ataques distribuídos de negação de serviço (*Distributed Denial of Service – DDoS*), além de roubar dados, enviar *spams* e utilizar a conexão do dispositivo afetado [Mahmoud et al. 2015]. Em todos estes casos, há uma coordenação entre os *bots* e uma influência do tráfego dos *bots* e seus alvos. Quanto maior a coordenação e maior a influência, maior a probabilidade dos nós pertencerem a uma *botnet* [Mirkovic et al. 2002, Ferreira and Nogueira 2018].

A detecção de *botnets* é um grande desafio, pois o conjunto de dados tratado tende a ser massivo. Um exemplo relativamente recente é o caso da *botnet* Mirai, que em 2016 infectou milhares de dispositivos da Internet das Coisas e desempenhou ataques DDoS cujos tráfegos atingiram a marca de 1.1 Tbps [Kolias et al. 2017]. Além disso, é preciso identificar quais características do tráfego são relevantes para distinguir dispositivos maliciosos de benignos. Diversos trabalhos propõem métodos baseados em aprendizagem de máquina para detectar uma *botnet*, tais como [Daya et al. 2019] e [Chowdhury et al. 2017]. Estes trabalhos se utilizam de características do tráfego, sem uma análise da relevância dessas características. Além disso, o volume de dados massivo a ser tratado torna essas propostas impraticáveis em um cenário real.

Dessa forma, este trabalho apresenta BotFetcher, um método híbrido de detecção de *botnets* que considera técnicas de agrupamento e processamento de sinais em grafos a fim de tratar a escala massiva dessas redes de *bots*. BotFetcher segue três etapas. A primeira realiza uma análise estatística das características do tráfego, buscando identificar quais são relevantes na análise daquele tráfego. A segunda utiliza as características selecionadas e uma técnica de aprendizagem de máquina não supervisionado para agrupar os dispositivos com comportamentos similares (*clustering*). Por fim, a terceira etapa identifica os *clusters* possuidores de nós com comportamento de *bots*, de modo a segregá-los dos outros nós. Além de viabilizar a análise, o agrupamento permite a paralelização da análise, acelerando a detecção.

A avaliação do método utilizou como entrada a base de dados CTU-13, oferecida pela Universidade da República Tcheca [García et al. 2014]. Esta base contém 13 cenários de captura de tráfego gerados por *botnets*. Neste trabalho, foram utilizados os cenários 5, 10 e 11. Como resultado, no cenário 5, o método reduziu o número de características selecionadas para análise e o efeito desta seleção na clusterização. Além disso, observaram-se a formação dos agrupamentos e o seu impacto na detecção dos *bots*. Por fim, a detecção foi avaliada em cada *cluster*, evidenciando os resultados da causalidade entre os nós. As métricas de avaliação são a acurácia do método e das etapas intermediárias (como a redução do número de características avaliadas, o número de *clusters* a serem avaliados até a detecção). Os resultados mostram que o método BotFetcher detectou o *bot* presente no cenário 5 da base CTU-13, com 55 falsos-positivos, dentre 39672 nós avaliados, não havendo falsos-negativos.

Este trabalho está organizado da seguinte forma. A Seção 2 discute os trabalhos relacionados e as pesquisas que fundamentam a proposta do método BotFetcher. A Seção 3 descreve o método de detecção de *botnets*, dividindo-o em três etapas. A Seção 4 descreve a metodologia de avaliação e os resultados. A Seção 5 conclui o trabalho.

2. Trabalhos Relacionados

Os primeiros estudos de detecção de *botnets* não se preocupavam com a quantidade de nós e como agrupá-los, pois o volume de tráfego trocado entre os dispositivos e a quantidade de dispositivos eram pequenos [Karasaridis et al. 2007]. O tráfego podia ser comparado com perfis específicos para identificar as *botnets*, como verificar o uso do IRC para comunicação [Binkley and Singh 2006]. O aumento na quantidade de tráfego e no volume de características geradas impulsionou estudos para identificar as características mais representativas, de modo a melhorar a detecção de *botnets*. Diversos métodos e técnicas

foram elaborados na área. Alguns dos trabalhos que possuem alguma intersecção com o BotFetcher são elencados, tais como: uso de grafos na análise, caracterização por tráfego e/ou grafos, seleção de características, clusterização e métodos de detecção de bots.

[Lagraa et al. 2017] realizaram uma análise na forma de grafos, formados pela sequência de dados trafegados pela rede. Os pacotes de tráfego de rede formam pequenos grafos dos quais se comparam suas formas e características, detectando os *bots* através de um comportamento pré-definido ou treinado. Recentemente, [Chowdhury et al. 2017] realizaram a detecção de *bots* através da análise de grafos formados através de características do tráfego. Os nós da rede são agrupados por similaridade de suas características no grafo. Assim, a maior parte dos nós não precisa ser analisada para encontrar os *bots*, sendo estes identificados em *clusters* menores quando comparado com o conjunto inicial de nós. [Daya et al. 2019] executaram a extração de características de grafos através do tráfego entre os nós. Porém, esse conjunto de características é fixo e não considera características da rede, apenas do grafo, não variando de acordo com o tráfego gerado. [Costa et al. 2018] apresenta uma técnica de detecção online de botnets através de mineração de fluxos. Este trabalho apresentou uma série de características de tráfego que caracterizam o tráfego. [Moussa et al. 2019] apresentaram os fundamentos teóricos do α-*investing*$^{+}$, um algoritmo para a análise estatística das características do tráfego. O objetivo foi identificar as características mais relevantes para a detecção de ataques DDoS. As características podem ser geradas diretamente do tráfego (como tamanho de pacote, endereços de origem e destino) ou a partir do tráfego (como as interações entre os nós presentes no tráfego). Porém, esse trabalho não focou na detecção de *bots*.

[Kohonen 2013] apresentou a técnica de aprendizagem de máquina não-supervisionada, *Self Organizing Map* (SOM), identificando dois dos principais algoritmos e formas de calibrá-los. No trabalho, foram apresentadas diversas áreas de aplicação para SOM, como a análise de dados exploratória. [Daya et al. 2019] utilizaram este algoritmo para o agrupamento de nós de rede e apresentaram melhores resultados quando comparado com outras técnicas de aprendizado de máquina não-supervisionado, tais como *k*-Means e DBScan. [Mei and Moura 2017] propuseram a técnica *Causal Graph Process*, que identifica inter-relações em séries temporais e intra-relações na série ao longo do tempo. Como resultado, é gerado um grafo representando as relações causais entre pares na série temporal. A técnica foi aplicada por [Ferreira and Nogueira 2018] para identificar *botnets* geradoras de ataques de *DDoS* volumétricos. Entretanto, a análise era feita em todo o conjunto de dados fornecido, não se preocupando em dividir o tráfego em *clusters* menores e nem realizar uma análise estatística para identificar quais características seriam importantes na análise. Com a identificação das variáveis relevantes, o conjunto de dados a ser analisado é reduzido. Além disso, a análise em *clusters* diminui a necessidade de processamento em um conjunto massivo de dados e propicia a paralelização da detecção.

3. O Método BotFetcher

Esta seção detalha o método BotFetcher, um método híbrido composto pelas as etapas de (i) extração e seleção de características, (ii) agrupamento (clusterização) e (iii) detecção por *cluster*, como ilustra a Figura 1. A etapa de extração e seleção de características serve para identificar aquelas mais relevantes a partir do tráfego de rede. Esta etapa tem como resultado um conjunto de características adequadas para a detecção de *bots*. A etapa de clusterização agrupa nós da rede com comportamentos similares no tráfego, cujas carac-

terísticas extraídas e selecionadas na etapa anterior se assemelham, de acordo com um processo de aprendizagem de máquina não-supervisionado. A etapa de detecção ocorre em paralelo em cada *cluster* através da análise causal das relações entre os nós.

BotFetcher é um método executado em um dispositivo computacional que tem como entrada características extraídas a partir dos cabeçalhos dos pacotes do tráfego (detalhadas na Subseção 3.1) ou, dependendo da implementação, os próprios cabeçalhos dos pacotes. Estes cabeçalhos são oriundos de sensores, monitores de rede ou roteadores capazes de processá-los e enviá-los ao BotFetcher. BotFetcher pode ou não pertencer a rede. A seguir, detalham-se os algoritmos e as técnicas aplicadas em cada etapa do método.

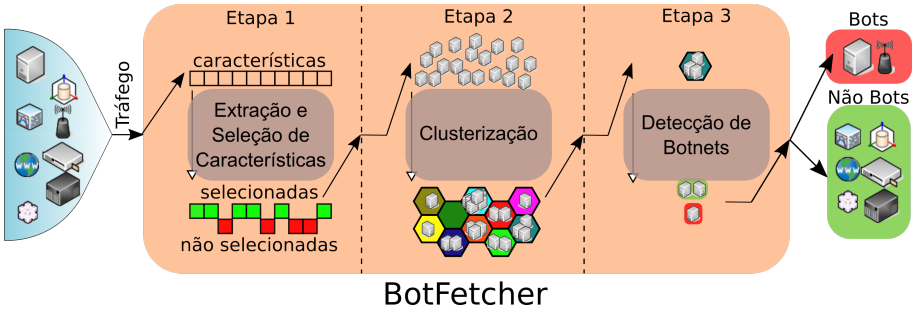

Figura 1. O método Botfetcher

3.1. Extração e seleção de características

Extrair e selecionar as características (*features*) utilizadas para a clusterização dos nós são essenciais para a eficiência na detecção dos *bots*. As características têm um papel fundamental no direcionamento da clusterização, sendo determinante para o funcionamento dos algoritmos de detecção. Neste etapa diversas características são extraídas e selecionadas do tráfego de rede e outras são formadas a partir do tráfego. As características podem ser coletadas tanto em roteadores de borda, de maneira descentralizada, ou em um roteador de núcleo, de maneira centralizada. Caso os roteadores de borda sejam utilizados, as características são extraídas no próprio dispositivo e enviadas ao BotFetcher que realizará as etapas subsequentes do método. Em ambos os casos, todo o tráfego é de conhecimento do BotFetcher para a realização da extração de características e análises posteriores. Dois tipos de características são extraídas de todo o tráfego: (i) características do tráfego propriamente dito e (ii) características que buscam evidenciar as relações entre os nós.

As características do tráfego são depreendidas do cabeçalho dos pacotes de rede. Elas estão sumarizadas na Tabela 1. A quantidade de protocolos contabiliza quantos protocolos cada nó utiliza para se comunicar com outros nós. Os protocolos ICMP, DNS, TCP e UDP são corriqueiros. O percentual do total de pacotes em que cada um destes protocolos aparece é contabilizado em *Percent* ICMP, *Percent* DNS, *Percent* TCP e *Percent* UDP, respectivamente. Certos ataques gerados por *botnets* utilizam prioritariamente destes protocolos, *e.g.,* o *Ping Flood* (protocolo ICMP) e Dyn Attack (protocolo DNS) e, portanto, estes percentuais são relevantes. Caso outros protocolos sejam utilizados, o percentual é contabilizado no parâmetro *percent* outros. Para as características *TCP Window Size* e *Frame Length*, as médias de seus valores para cada nó são calculadas, pois podem apresentar características diferentes de outros nós em caso de comunicação por *bot*. A janela TCP indica a quantidade de *bytes* enviados antes de um *ACK*. O tamanho

Características	Descrição
Quantidade de Protocolos	Número de protocolos que um nó se comunica
TTL Médio	*Time to live* médio dos pacotes enviados
TCP Window Size	Tamanho da janela no protocolo TCP
Percent TCP	Percentagem de pacotes que utilizam protocolo TCP
Percent UDP	Percentagem de pacotes que utilizam protocolo UDP
Percent DNS	Percentagem de pacotes que utilizam protocolo DNS
Percent ICMP	Percentagem de pacotes que utilizam protocolo ICMP
Percent Outros	Percentagem de pacotes que utilizam outros protocolos
Source Privileged Ports	Quantidade de portas de origem que são privilegiadas ($<$1024)
Source Not Privileged Ports	Quantidade de portas de origem que não são privilegiadas ($>$1024)
Quantidade de Portas de Destino	Quantidade de portas de destino de pacotes enviados por um nó
Frame Length	Tamanho do quadro do pacote
In Degree	Caracerística de grafo definida no texto
Out Degree	Caracerística de grafo definida no texto
In Degree Weight	Caracerística de grafo definida no texto
Out Degree Weight	Caracerística de grafo definida no texto
Node betweenness centrality	Caracerística de grafo definida no texto
Local clustering coefficient	Caracerística de grafo definida no texto
Eigenvector centrality	Caracerística de grafo definida no texto

Tabela 1. Características extraídas do tráfego para cada nó

do quadro se refere ao quadro da camada de enlace. Certos tipos de *bots* se beneficiam de privilégios no dispositivo infectado e enviam pacotes a partir de portas privilegiadas (menores que 1024), enquanto outros só conseguem enviar a partir de portas não privilegiadas (maiores que 1024). As características *Source Privileged Ports* e *Source Not Privileged Ports* contabilizam a quantidade de pacotes que tem origem em portas privilegiadas e não privilegiadas, respectivamente. Também, a quantidade de portas de destino é contabilizada para discriminar *bots* de outros nós, pois alguns *bots* se comunicam prioritariamente com uma determinada porta de destino. A média do valor de TTL (*Time To Live*) do pacote é calculada pois os *bots* podem tentar se esconder enviando pacotes com TTLs muito pequenos ou muito grandes.

Além das características do tráfego, outras são extraídas buscando identificar as relações entre os nós. BotFetcher cria um grafo direcionado de modo a evidenciar essas interações. A cada pacote trocado entre o nó origem e o nó destino, os endereços IP de origem e destino são capturados e serão vértices do grafo construído pelo BotFetcher. Uma aresta direcionada é criada do nó de origem para o nó de destino com base nos endereços IP de origem e destino indicados em cada pacote. Havendo uma aresta existente entre as mesmas origem e destino, outra aresta não será adicionada. Os endereços de origem e destino dos pacotes enviados e recebidos são capturados e enviadas ao BotFetcher que realizará a construção de um grafo G_1=(V,Ar), onde V é o conjunto de endereços IP e Ar, conjunto de arestas indicadas pelo par ordenado endereço IP de origem e endereço IP de destino de um pacote do tráfego. Cabe ao BotFetcher construir uma visão geral da rede através do grafo. Assim, as seguintes características são extraídas do grafo para buscar caracterizar o comportamento do nó: *out degree*, *in degree*, *out degree weight*, *in degree weight*, *local clustering coefficient*, *node betweenness centrality* e *eigenvector centrality*.

In degree e *out degree* são a quantidade de nós que se comunicam com um dado nó. *In degree* indica a quantidade de nós que enviaram ao menos um pacote para o dado

nó, enquanto *out degree* indica a quantidade nós em que o nó enviou ao menos um pacote de tráfego de rede. *In degree* e *out degree* são relevantes, pois se muitos *bots* tentam contactar um único nó atuando como *botmaster*, pode ter um alto valor de *in degree* para o *botmaster*. *Out degree weight* e *in degree weight* são similares às características anteriores, se diferenciando, porém, por contabilizar a quantidade de pacotes trocados de e para um dado nó, respectivamente. A cada pacote trocado, havendo uma aresta existente com a mesma origem e destino, a quantidade de um pacote é somada ao valor atual, senão a aresta é criada e estas características são inicializadas. *Node betweenness centrality* quantifica o número de vezes que um nó atua como elo para o menor caminho entre dois outros nós. Quanto maior o valor do *node betweenness centrality*, mais central no grafo um nó é, podendo ser relevante para a detecção de *botnets* em que os nós se comunicam em uma estrutura descentralizada, como uma uma rede de *bots* P2P [Chowdhury et al. 2017]. *Local clustering coefficient* indica quão concentrada é a vizinhança de um dado nó, buscando identificar quão próximos no grafo os vizinhos de um nó estão. Por fim, *eigenvector centrality* é um critério que busca evidenciar a influência de um nó sobre outros, indicando o peso normalizado de um nó, no intervalo entre 0 e 1.

Todas as 19 características são normalizadas, o que é importante para a etapa de clusterização. Entretanto, elas podem ou não ser relevantes para a análise. Embora ter mais características pareça interessante, em aprendizado de máquina (utilizado na etapa de clusterização – detalhes na Subseção 3.2) isto pode incorrer na Maldição da Dimensionalidade, como descrito em [Bellman 1966]. Isto indica a existência de um limite para melhorias no resultado decorrente do aumento no número de características. Assim, um conjunto estatisticamente relevante de características deve ser buscado para as análises subsequentes. Este é o objetivo da seleção de características, que busca pelas características mais relevantes e informativas em relação aos dados. Neste trabalho, o algoritmo α-*investing*$^+$ [Moussa et al. 2019] (Algoritmo 1) realiza a seleção de características. Este algoritmo objetiva reduzir a taxa de falsos positivos na descoberta de novas características. Um falso positivo, nesta etapa, consiste em uma característica recém-descoberta, erroneamente selecionada e adicionada ao conjunto de características relevantes até um dado momento. No α-*investing*$^+$, uma característica é adicionada ao conjunto de características selecionadas se ela não tiver relação de dependência com alguma característica pertencente ao conjunto. Para tanto, um limiar α ($l.5$ no Algoritmo 1) determina a probabilidade de incluir uma característica irrelevante ao conjunto. Além disso, um peso w ($l.2$) é atribuído representando o número de possíveis falsos positivos que podem ser incluídos ao conjunto ao longo das iterações do algoritmo. Um falso positivo, então, é uma característica com alguma relação de dependência com outra no conjunto de características selecionadas. O valor p é a probabilidade de duas características serem independentes e resulta de um teste estatístico. No caso do α-*investing*$^+$, o teste é o Chi-Quadrado.

Dessa forma, para adicionar uma nova característica ao conjunto de características selecionadas, é necessário que o valor do teste estatístico associado a esta característica seja maior que o limiar α atual ($l.13$ a $l.18$). Neste caso, o peso w atual aumenta ($l.17$) e, no caso contrário, onde a característica é dependente de outra, w diminui ($l.18$). O valor de w afeta diretamente o limiar α ($l.9$), de forma que quanto mais características são adicionadas ao conjunto de características selecionadas, o limiar torna-se mais rígido para impedir a admissão de falsos positivos. Para cada uma das 19 características extraídas, um vetor (denominado vetor de características) com os valores das características

Algoritmo 1: Algoritmo α-*investing*$^+$ (extraído de [Moussa et al. 2019])

Dados: Logs de rede
Resultado: Conjunto de características selecionadas

$W \leftarrow [0.5]$; // Conjunto de pesos
$w_0 \leftarrow W_0$; // Peso inicial
$F \leftarrow \{\}$; // Conjunto de características detectadas
$S \leftarrow \{\}$; // Conjunto de características selecionadas
$\alpha_\Delta \leftarrow 0.5$;
i $\leftarrow 1$;
enquanto *novas características forem detectadas* **faça**
    adicionarFeature(f_i, F);
    $\alpha_i \leftarrow w_i/2 * i$;
    para *cada* $s \in S$ *tal que* $s \neq f_i$ **faça**
        tabela $\leftarrow$ gerarTabelaContingencia(f_i, m);
        resultado $\leftarrow$ chiQuadrado(tabela);
        se *resultado[valorP]* $\leq \alpha_i$ **então**
            $w_{i+1} \leftarrow w_i - \alpha_i$;
        senão
            adicionarFeature(f_i, S);
            $w_{i+1} \leftarrow w_i + \alpha_\Delta - \alpha_i$;
        fim
    fim
    i $\leftarrow$ i + 1;
fim

é formado para cada nó. O algoritmo tem como entradas os vetores de características e é executado até que não haja detecção de novas características. No caso do método Botfetcher, são avaliadas as 19 características continuamente geradas. O α-*investing*$^+$ oferece uma redução do conjunto de dados a ser analisado, melhorando o desempenho do método, sem que a representatividade dos dados seja comprometida. A ordem de entrada das características afeta o resultado do algoritmo. Preferencialmente, inicia-se a análise com as características com menor custo computacional para sua extração. Ao fim da execução, obtém-se um conjunto de características que melhor representam o conjunto de dados.

3.2. Clusterização

Nesta etapa, o método BotFetcher utiliza uma técnica de aprendizagem de máquina não-supervisionada para agrupamento dos nós, particularmente, a técnica Self-Organizing Map (SOM). Esta técnica oferece uma redução de dimensionalidade nos dados e segue uma fase de treinamento, indicando que não conhecem *a priori* a classificação das entradas. Nesta etapa, portanto, o objetivo é identificar como os nós devem ser agrupados. Os nós são agrupados analisando as características selecionadas no procedimento descrito na Subseção 3.1. Desta forma, cada nó possui um vetor de características como entrada para a clusterização. A técnica SOM compara esse vetor de características com o de outros nós, através do cálculo da distância euclidiana entre os vetores. A distância euclidiana é calcula pela raiz quadrada da soma dos quadrados das diferenças dos termos dos vetores (Equação 1). Os termos dos vetores são os valores das características selecionadas e o centro do *cluster*. O nó é atribuído ao *cluster* cujo seu vetor de características seja

mais próximo do centro daquele *cluster*. Dados os vetores Q_1 = (q_1,q_2,...,q_{19}) e S_1 = (s_1,s_2,...,s_{19}), a distância euclidiana é cálculada através da Equação 1.

$$d_E(Q,S) = \sqrt{(q_1 - s_1)^2 + (q_2 - s_2)^2 + \cdots + (q_{19} - s_{19})^2} = \sqrt{\sum_{i=1}^{n}(q_i - s_i)^2} \qquad (1)$$

As distâncias euclidianas entre estes vetores são mensuradas para associar um nó a um *cluster*. Calculadas as distâncias, o SOM retornará um *grid* 5x5, identificado no código como um *array* 5x5. Cada entrada deste *array* corresponde a um *cluster* e com os valores dos endereços IP dos nós associados àquele *cluster* (uma entrada do *array*). Um *cluster* é uma coleção de nós similares. A cada iteração, os centros dos *clusters* são ajustados de modo que as entradas contenham a maior quantidade de nós com a menor distância euclidiana para o centro do seu *cluster*. As iterações ocorrem até que o centro dos *clusters* não seja mais alterado (e portanto os nós em cada *cluster* ou até um limite de iterações ser atingido (na implementação descrita a seguir, o limite de iterações é 1000).

Com a clusterização, a dimensão da análise passa da granularidade de nós para a de *clusters*. Reduzindo significativamente o espaço de análise, com um conjunto muito menor de nós em cada *cluster*. Assim, a vasta distribuição de dados de entrada é rearranjada em um conjunto finito de *clusters*. Os *clusters* são distribuídos em uma grade (*grid*) que possibilita compreender as relações entre os nós, especialmente quando se trata de um conjunto bastante volumoso de informações, como é o caso da análise de tráfego de rede. No BotFetcher, os nós serão agrupados em uma grade 5x5 (totalizando 25 *clusters*, utilizando-se as características de rede extraídas e selecionadas para identificar as similaridades, ou seja, nós cujos vetores de características pertençam a um mesmo *cluster*.

É esperado que alguns *clusters* contenham um número maior de nós do que os outros. Esses com maior número de nós representam o conjunto de nós com comportamento benigno na rede [Chowdhury et al. 2017]. Comportamentos anormais/maliciosos são pouco frequentes no mundo real. Na base de dados CTU-13, cenário 5, mais de 99% dos nós não são infectados por *bots* (e em outros cenários desta mesma base, temos dados similares). Modelos simulados indicam até 50% de nós que podem ser infectados [Lan et al. 2009]. A infecção, também, não é imediata e de imediata ação pelos nós infectados, ocorrendo ao longo do tempo. No caso do *ransomware* Wannacry, que infectou rapidamente diversas empresas e máquinas ao redor do globo, estima-se até 30% dos dispositivos que tinha a porta 445 TCP aberta a Internet (meio de infecção dos dispositivos) eram vulneráveis [Rapid7 2017]. Esta premissa permite que inicie-se a análise pelo menor *cluster*, acelerando a detecção dos *bots*, com um custo computacional muito menor do que realizar a análise sobre todo o tráfego. Parte-se da detecção no menor *cluster*, seguido pelo segundo menor, até se esgotar toda a análise.

3.3. Detecção de *bots*

Para a detecção de *bots* busca-se correlacionar o tráfego entre os nós e compreender as interrelações entre os dados coletados e suas intrarelações ao longo do tempo, através de técnicas de processamento de sinais em grafos. Nota-se que estas técnicas não consistem na área de processamento de sinais, apesar de estarem fundamentadas na mesma. Neste

processo, utiliza-se autoregressão sobre séries temporais. Para cada nó e característica, tem-se uma série temporal, ou seja, uma coleção de valores distribuídos em intervalos de tempo regulares, do início do período de avaliação até seu fim. Os valores coletados ao longo do tempo são divididos em janelas de tempo regularmente espaçadas, de modo que o tamanho da amostra temporal é o mesmo para todos os nós avaliados e as as séries contenham a mesma quantidade de dados. Das 19 características extraídas, cada uma possui uma série temporal correspondente. Porém, neste trabalho utilizamos apenas a característica *out degree weight* (quantidade de pacotes) com base em resultados prévios [Ferreira and Nogueira 2018]. Uma matriz de influências é a saída deste processo, possibilitando identificar as relações causais do tráfego gerado pelos nós e permitindo identificar o tráfego de *bots*. Cada entrada da matriz corresponde ao peso atribuído à relação entre os dois nós identificados na linha e na coluna da matriz.

Definidas as séries temporais, a autoregressão é aplicada e estimará a matriz de influências. Os coeficientes da autoregressão são filtros que permitem reduzir ainda mais o espaço de busca dos dados [Sandryhaila and Moura 2013]. A matriz de influências permite capturar as relações causais entre os nós e suas correlações [Mei and Moura 2017]. Calcula-se o grau de influência entre os nós identificados no tráfego desconhecendo *a priori* a relação entre os dados. Porém, uma característica em uma janela de tempo *k* é influenciada de alguma forma pela característica em uma janela de tempo *k*-1. Esta é a influência que a autoregressão identifica, explicitada na matriz. A Equação 2 descreve o detalhamento matemático para determinar a matriz de influências [Sandryhaila and Moura 2013].

$$
\begin{aligned}
x\left[k\right] &= w\left[k\right] + \sum_{i=1}^{M} Pi(\mathbf{A}, \mathbf{c}) x\left[k-1\right] \\
&= w\left[k\right] + \sum_{i=1}^{M} \sum_{j=0}^{i} (c_{ij}\mathbf{A}^{j}) x\left[k-1\right] \\
&= w\left[k\right] + (c_{10}\mathbf{I} + c_{11}\mathbf{A}) x\left[k-1\right] + \\
&(c_{20}\mathbf{I} + c_{21}\mathbf{A} + c_{22}\mathbf{A}^{2}) x\left[k-2\right] + ... \\
&+ (c_{M0}\mathbf{I} + ... + c_{MM}\mathbf{A}^{M}) x\left[k-M\right]
\end{aligned}
\tag{2}
$$

onde x*[k]* é o valor da característica no tempo k, *Pi*(**A**, **c**) é um polinômio da matriz em A de ordem *i* (isto é, *Pi*(**A**, **c**) são filtros [Sandryhaila and Moura 2013]); w*[k]* é ruído estatístico, utilizado na avaliação de precisão da autoregressão; c_{ij} são coeficientes polinomiais escalares, sendo **c** = $(c_{10}\ c_{11}...c_{ij}...c_{MM})^{(T)}$ um vetor de todos os c_{ij}, e *M* é a ordem da autoregressão [Mei and Moura 2017]. Todos os termos são conhecidos, exceto a matriz de influências **A**. Ao final deste processo, os graus de correlação entre os nós são identificados na matriz de adjacências. Quanto maior a coordenação entre os nós, ou seja, quanto maior o grau de correlação entre eles, maior a possibilidade de ser identificada uma *botnet*. A coordenação entre os nós é indicada pela magnitude dos valores na matriz de adjacência. Quanto mais distante do zero, maior é a influência entre os nós identificados na linha e na coluna da matriz.

No BotFetcher, a detecção é realizada em cada *cluster*. Assim, o volume de dados é menor do que em [Ferreira and Nogueira 2018]. Para cada *cluster*, as relações entre

os nós do *cluster* podem ser calculadas, tendo como saída uma matriz *N*x*N*, onde *N* é número de nós do *cluster*. Calculadas as matrizes de influências para cada *cluster*, os *bots* terão valores de ordens de magnitude maior que os nós que não apresentam nenhuma correlação entre si. Por realizar a detecção em *clusters*, o BotFetcher pode paralelizar a análise, trazendo ganhos de desempenho para o método. As matrizes são avaliadas individualmente para cada *cluster*, identificando os *bots* em cada *cluster*. O BotFetcher classificará os nós em *bots* ou não. Dessa forma, os resultados podem ser informados após a análise de cada *cluster*, para atuação imediata ou após a análise de todos os *clusters*.

4. Avaliação

Esta seção descreve a metodologia de avaliação do método proposto. O objetivo principal do método é aumentar a acurácia na detecção de *botnets*. Assim, o método é avaliado sob esta métrica. Como as etapas do BotFetcher funcionam independentemente e com limites e atribuições claramente definidos, avalia-se também cada etapa separadamente. Neste sentido, apresentam-se os cenários e os resultados para todas as etapas.

O tráfego utilizado na avaliação é da base de dados CTU-13. Essa base foi gerada, em um ambiente virtualizado, pela Universidade da República Tcheca, com 13 cenários diferentes de *botnets*. A máquina física utilizada por eles tinha o Debian como sistema operacional e as virtualizações com Windows XP. Ao menos uma máquina virtualizada, em todos os cenários, estava infectada com um *malware* que gerava tráfego característico de uma *botnet*. As demais máquinas virtuais possuíam tráfego normal. Assim, na base de dados, tem-se o tráfego normal, o da *botnet* e um tráfego de fundo.

Inicialmente, os dados de captura da base de dados CTU-13 são tratados pela etapa de extração e seleção de características. O método seleciona então as características mais relevantes para a análise. Para a clusterização, apresentamos os cenários com a utilização da seleção de características e sem a sua utilização. Indicamos qual o *cluster* possui o *bot*. Além disso, mostra-se a redução alcançada por esta análise, partindo do menor *cluster* até o maior, até a detecção dos *bots*. Por fim, para detecção dos *bots*, avaliam-se as matrizes de influência para cada *cluster*, de modo a evidenciar a magnitude dos valores na matriz em um *cluster* com a presença do *bot* e sem a sua presença.

Executamos os testes em uma máquina com processador Intel(R) Xeon(R) Silver 4114 CPU@2.20GHz, com 64 GB de RAM e sistema operacional Ubuntu Server 18.04.3 LTS. Para a extração dos dados e cálculo de características, foi utilizado Python, com a bibilioteca NetworkX. As implementações para a seleção de características foram feitas em Python e R. Para a clusterização, Python foi a linguagem utilizada com a bibilioteca MiniSOM. Por fim, para a detecção de *bots*, a implementação foi feita em objetice-C (arquivo *.m) que pode ser executado tanto em Matlab e Octave.

Resultados

Cenários comumente analisados em detecção de *botnets* são os cenários 10 e 11 da base de dados CTU-13. Assim, nós também iniciamos analisando estes cenários. O cenário 11 possui três *bots* com o *malware* Rbot. Entretanto, possui apenas outros 10 nós, totalizando 13 nós. O resultado da clusterização mostrou poucos *clusters* povoados e muitos vazios, como pode ser visto na Tabela 4, representando o *grid* para este cenário. Note o *grid* 5x5, com apenas 8 *clusters* contendo nós, enquanto outros 17 *clusters* estão vazios.

	0	1	2	3	4
0	**2**	4	0	1	0
1	0	1	1	1	0
2	0	**1**	4	0	0
3	0	0	0	0	0
4	0	0	0	0	0

Tabela 2. Cenário 11/CTU-13: *clusters*: Dois bots em (0,0) e um bot em (2,1)

Diante deste resultado, poderia-se reduzir o número de *clusters* para 2, por exemplo, para executar o método BotFetcher por inteiro. Entretanto, com base nos resultados apresentados em [Ferreira and Nogueira 2018], os autores mostraram que métodos mais simples sem clusterizações são capazes de detectar os *bots* em cenário similar. O cenário 10 apresenta também poucos nós: 16, no total, sendo 10 nós com bots e 6 nós com comportamento normal. Nesse sentido, utilizamos o cenário 5 da base de dados CTU-13. Esta base contém apenas um nó infectado com o *malware* Virut. Este *malware* é utilizado para ataques DDoS, spam, fraude, roubo de dados e instalações maliciosas. No terceiro trimestre de 2012, foi responsável por mais de 5,5% das infecções de computadores. Neste cenário da CTU-13, tem-se o tráfego de 39738 nós. O *bot* realiza o ataque identificando *proxies*, escaneando portas e utilizando-os, quando vulneráveis [Symantec 2013].

A Tabela 3 indica as características selecionadas na primeira etapa do BotFetcher para o cenário 5 da base CTU-13 e aquelas características excluídas. Obteve-se uma redução de 48% no conjunto total de características. Vale ressaltar que uma das características não selecionadas, a *node betweenness centrality* é a que tem maior custo computacional no cálculo das características do grafo gerado. Embora não haja maneira de saber antes da execução que esta característica não seria selecionada, em alguns casos, 99,9% do tempo foi gasto com esta característica. Em todos os testes, o tempo de cálculo sempre ficou acima de 95% (resultado similar ao encontrado em [Daya et al. 2019]).

Excluída	**Selecionada**
TCP Windows Size	TTL Médio
Percent DNS	Quantidade de Portas de Destino
Percent UDP	Frame Length
Source Privileged Ports	Percent ICMP
Eigenvector Centrality	Percent TCP
Out Degree Weight	Percent Outros
Out Degree	Source Not Privileged Ports
Node Betweenness Centrality	In Degree Weight
Quantidade de Protocolos	In Degree
	Local clustering coefficient

Tabela 3. Cenário 5/CTU-13: Características excluídas e selecionadas

Para a clusterização, identificam-se 25 *clusters*. Utilizando as características selecionadas, o *cluster* com menor quantidade de nós identificou para si 47 nós de rede (marcado na cor azul na Tabela 4). O *cluster* com o nó infectado ficou identificado no segundo menor *cluster* com outros 75 nós (marcado na cor vermelha na Tabela 4). O maior *cluster*, com apenas tráfego normal e ordinário, possui 6934 nós. Até alcançar o *cluster*

	0	1	2	3	4
0	99	**75**	125	444	3540
1	1121	47	251	1009	202
2	728	2484	656	627	5602
3	345	459	2401	1052	4851
4	287	1137	1226	6934	4035

Tabela 4. *Clusters* com as características selecionadas

	0	1	2	3	4
0	21567	667	160	110	444
1	3257	77	241	98	77
2	5190	431	593	212	**716**
3	1282	160	331	28	114
4	2841	1007	84	38	12

Tabela 5. *Clusters* com a utilização das 19 características extraídas

com o *bot*, teria-se que analisar somente o *cluster* com 47 nós. Vale notar, também, que a vizinhança ao redor do *cluster* com o *bot* possui os *clusters* com menor quantidade de nós. Evidencia-se a vantagem desta técnica em trazer para perto os *clusters* com comportamento parecido. Como esta técnica tem um componente aleatório na sua inicialização, diversas execuções foram feitas sempre com resultados similares.

Quando não é realizada a seleção de características e o conjunto das 19 características extraídas a partir do tráfego é utilizada, tem-se *clusters* com maior segregação. Os menores *clusters* tem menos nós do que no cenário com a seleção de características e o maior *cluster* tem mais de três vezes o tamanho do *cluster* com mais nós quando aplicada a seleção de características. Além disso, sem a seleção, tem-se diversos *clusters* com menor quantidade de nós até chegar ao *cluster* com o *bot* Virut, como pode ser visto na Tabela 5. Dentre esses, 18 *clusters* com menos nós podem ser identificados entre o menor *cluster* e o *cluster* com o *bot*. Como esta técnica tem um componente aleatório na sua inicialização, diversas execuções foram feitas, também neste cenário, sempre obtendo resultados similares. Fica evidenciada, então, a vantagem neste cenário da aplicação da técnica de seleção de características comparada com sua não aplicação.

A detecção dos *bots* resulta da análise da matriz de influência de cada *cluster*. A análise inicia no menor *cluster*. Com a matriz de influências, pode-se verificar a existência ou não influências entre os nós, utilizando as características de total de pacotes trocados entre os nós e soma dos tamanhos dos pacotes trocados. Quando ná correlação entre os nós, diversos *clusters* apresentam valores baixos na matriz. Na Tabela 6, têm-se as influências entre os nós com valores de magnitude maiores que na Tabela 7, onde não há *bot*. Porém, dentro deste *cluster*, há falsos-positivos, onde ocorre uma grande influência entre os nós, sem que estes sejam de fato *bots*. Dos 75 nós deste *cluster*, 41 nós não têm influência alguma uns sobre os outros, enquanto 34 nós têm grande influência entre si.

Além disso, a análise mostrou nós com influência entre si no menor *cluster*. Dos 47 nós deste *cluster*, 22 nós possuem influência entre si, enquanto outros 25 não possuíam correlação alguma. Para os demais *clusters*, não foi identificada correlação significativa entre os nós. Portanto, os demais 39616 nós foram corretamente identificados como be-

IPs dos nós	41.143.58.184	147.32.87.36	207.200.96.138	195.34.208.134	...	112.210.209.230	194.108.204.18	**147.32.84.165**
41.143.58.184	2.63e+4	2.25e+4	0,0000	0,0000	...	-1.06e+4	0,0000	2.50e+4
147.32.87.36	-1.28e+3	-2.05e+4	0,0000	0,0000	...	1.34e+4	0,0000	-1.00e+5
207.200.96.138	0,0000	0,0000	0,0000	0,0000	...	0	0,0000	0
195.34.208.134	0,0000	0,0000	0,0000	0,0000	...	0	0,0000	0
...	...	...	...	...	...			
147.32.84.165	1.43e+3	4.10e+3	0	0	...	-2.13e+3	0	-1.32e+4

Tabela 6. Matriz de Influências em *cluster* com o *bot* Virut

IPs dos nós	111.249.15.31	126.26.46.73	87.50.31.71	...	80.203.98.78	86.174.128.34	81.91.210.3	92.47.203.83
14.97.35.197	-0,033	0,167	0,000	...	0,000	0,000	0,000	0,000
208.54.45.165	0,161	0,118	0,000	...	0,000	0,000	0,000	0,000
186.253.91.47	0,000	0,000	0,000	...	0,000	0,000	0,000	0,000
213.81.131.80	0,000	0,000	0,000	...	0,000	-0,005	0,000	0,000
...	...	...	...	...	...	...	...	...
89.192.0.14	0,000	0,000	0,000	...	0,000	0,000	0,035	0,310
121.54.54.44	0,000	0,000	0,000	...	0,000	0,000	0,000	0,074
78.144.229.1	0,000	0,000	0,000	...	0,000	0,000	0,000	0,000

Tabela 7. Matriz de Influências em *cluster* sem *bots*

nignos, não havendo falso-negativo (Tabela 8). Caso a detecção de *bots* se desse sobre toda a base de dados, o tempo de execução para a análise se tornaria impraticável. Em *clusters*, a análise é acelerada pois um nó de um *cluster* não será comparado para identificar influências com um nó de outro *cluster*. Essa separação permite também a paralelização da análise, podendo cada *cluster* ser analisado em um dispositivo diferente, coordenado pelo BotFetcher

		Real	
		Bot	Benigno
Predição	Bot	1	55
	Benigno	0	39616

Tabela 8. Matriz de confusão gerada após o algoritmo de detecção de *bots*

5. Conclusão

Este trabalho apresentou BotFetcher, um método de detecção de *botnets* proposto para operar com um grande volume de dados de análise. O uso de características híbridas, provenientes de análise de grafos e do tráfego aumentou as informações que possibilitam a detecção dos *bots*. Com a seleção de características, tem-se uma redução no volume de dados a serem analisados, sem perder qualidade na análise como pode ser verificado nos resultados. A clusterização propicia uma análise mais eficaz pois parte dos menores *clusters* até chegar ao maior, podendo identificar um *bot* mais rapidamente e mitigando seus efeitos. Por fim, a análise de cada *cluster* pode ser paralelizada em diversos dispositivos, acelerando os resultados. O cenário testado identificou o *bot* presente na base, entre outros 39672 nós avaliados, com apenas 55 falsos-positivos e sem falsos-negativos.

Referências

Bellman, R. (1966). Dynamic Programming. *Science*, 153(3731):34–37.

Binkley, J. R. and Singh, S. (2006). An algorithm for anomaly-based botnet detection. In *Conference on Steps to Reducing Unwanted Traffic on the Internet*, Berkeley, CA.

Chowdhury, S., Khanzadeh, M., Akula, R., Zhang, F., Zhang, S., Medal, H., Marufuzzaman, M., and Bian, L. (2017). Botnet detection using graph-based feature clustering. *Journal of Big Data*, 4(1):1–14.

Cisco (2018). Visual Networking Index Complete Forecast Highlights. Disponível em: https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_Device_Growth_Traffic_Profiles.pdf. Acessado em 22/12/2019.

Costa, V. G. T. d., Zarpelão, B. B., Miani, R. S., and Junior, S. B. (2018). Online detection of botnets on network flows using stream mining. SBC.

Daya, A. A., Salahuddin, M. A., Limam, N., and Boutaba, R. (2019). A Graph-Based Machine Learning Approach for Bot Detection. *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 144–152.

Ferreira, A. E. G. and Nogueira, M. (2018). Identificando botnets geradoras de ataques ddos volumétricos por processamento de sinais em grafos. In *Anais do Workshop de Gerência e Operação de Redes e Serviços*, Porto Alegre, RS, Brasil. SBC.

García, S., Grill, M., Stiborek, J., and Zunino, A. (2014). An empirical comparison of botnet detection methods. *Computers & Security*, 45:100–123.

Karasaridis, A., Rexroad, B., and Hoeflin, D. (2007). Wide-scale botnet detection and characterization. In *Conference on First Workshop on Hot Topics in Understanding Botnets*, pages 1–7, Berkeley, CA, USA. USENIX Association.

Kohonen, T. (2013). Essentials of the self-organizing map. *Neural Networks*, 37:52–65.

Kolias, C., Kambourakis, G., Stavrou, A., and Voas, J. (2017). DDoS in the IoT: Mirai and other botnets. *Computer*, 50(7):80–84.

Lagraa, S., Francois, J., Lahmadi, A., Miner, M., Hammerschmidt, C., and State, R. (2017). Botgm: Unsupervised graph mining to detect botnets in traffic flows. In *Cyber Security in Networking Conference (CSNet)*. IEEE.

Lan, K.-C., Hussain, A., and Dutta, D. (2009). The effect of malicious traffic on the network. *Proc. Passive and Active Measurement Wksp. (PAM)*.

Mahmoud, M., Nir, M., and Matrawy, A. (2015). A Survey on botnet architectures, detection and defences. *International Journal of Network Security*, 17(3):272–289.

Mei, J. and Moura, J. M. F. (2017). Signal processing on graphs: Causal modeling of unstructured data. *IEEE Transactions on Signal Processing*, 65(8):2077–2092.

Miniwatts Marketing Group (2019). Internet Usage Statistics - World Internet Users and 2019 Population Stats. Disponível em: https://www.internetworldstats.com/stats.htm. Acessado em 02/12/2019.

Mirkovic, J., Prier, G., and Reiher, P. (2002). Attacking ddos at the source. In *IEEE International Conference on Network Protocols, 2002.*, pages 312–321.

Moussa, A. A., Nogueira, M., and Guedes, A. L. (2019). Seleção Online de Features em Streaming Baseada em Alpha-investing para Ataques DDoS. In *WGRS*. SBC.

Rapid7 (2017). WannaCry Update: Vulnerable SMB Shares Are Widely Deployed And People Are Scanning For Them (Port 445 Exploit). Acessado em 24/12/2019.

Sandryhaila, A. and Moura, J. M. F. (2013). Discrete signal processing on graphs. *IEEE Transactions on Signal Processing*, 61(7):1644–1656.

Symantec (2013). Snapshot of Virut Botnet After Interruption. Disponível em: https://www.symantec.com/connect/blogs/snapshot-virut-botnet-after-interruption. Acessado em 18/12/2019.