Detecção de tráfego anômalo de rede utilizando clusterização em Big Data

Mateus Rocha e Daniel G. Silva

***Resumo*— Dado o elevado e crescente tráfego de informações pela Internet, é possível empregar ferramentas de Big Data em associação a modelos de aprendizado de máquina para se detectar tráfego malicioso em redes de computadores. Daí, dois grandes desafios emergem nesse contexto: a elevada latência para se processar grandes volumes de dados e a menor flexibilidade que o paradigma de aprendizado supervisionado pode trazer. Neste trabalho, propõe-se uma arquitetura em Big Data que adota o paradigma não-supervisionado para detecção de intrusão em *near real time* e que também permite o retreino periódico dos modelos gerados, a fim de se ajustar às constantes mudanças de rede. A arquitetura desenvolvida é testada com sucesso em um conjunto de dados reconhecido para o problema e em tráfego de rede real de um servidor de Proxy Reverso.**

***Palavras-Chave*— Big Data, clusterização, detecção de intrusão em rede**

***Abstract*— Nowadays, the sheer amount of information sent through the Internet enables the adoption of Big Data and machine learning frameworks in order to detect network anomalies. However, there are two key challenges: the processing latency due to huge amounts of data and the reduced flexibility that supervised learning paradigm may cause. In this paper, we propose a Big Data framework that uses unsupervised learning for near real time intrusion detection, which is also capable of periodically retrain the generated models in order to track the network dynamics. The framework is succesfully tested with a well-known dataset and with real network traffic from a reverse proxy server.**

***Keywords*— Big data, clustering, network intrusion detection.**

I. Introdução

O número de ataques cibernéticos continua a crescer atualmente. As ameaças virtuais influenciam os cenários social, econômico e político do mundo moderno, através de assaltos internacionais a bancos, influência em eleições políticas e até mesmo governos de países patrocinando ataques cibernéticos [1]. Dentro desta realidade, sistemas de detecção de intrusão podem ser úteis na (*i*) identificação de atividades suspeitas e/ou não autorizadas que comprometam a segurança do ambiente e (*ii*) na tomada de medidas reativas para minimizar os riscos. No entanto, pode ser necessário uma etapa prévia de treinamento desses sistemas e, no contexto de Big Data, em que se pode explorar grandes volumes de tráfego de rede para gerar modelos de detecção de atividades suspeitas, surge o desafio de processar tal volume massivo de informação em tempo ágil.

Há uma série de trabalhos sobre detecção de intrusão e tráfego anômalo, associado (ou não) a Big Data. Bloedorn et al. [2] provém uma introdução sobre mineração de dados para detecção de intrusão em sistemas. Tal discussão inicial é complementada em [3], onde diferentes *features* são listadas e analisadas. Já o estudo em [4] analisa diversas técnicas para detecção de anomalias de rede à luz de fatores como escalabilidade e complexidade computacional dos algoritmos.

Zhong et al. [5] focam na utilização de clusterização para detectar intrusão em tráfego de rede, com a proposição de um algoritmo específico que, porém, não lida com dados de natureza categórica. Já Wang et al. [6] revisam o uso de *Big Data Analytics* para detecção de intrusão de rede; diversos pontos importantes são listados, incluindo algoritmos para detecção, a possibilidade de usar aprendizado de máquina e *frameworks* para processamento de dados.

Ainda no contexto de aprendizado de máquina, Suthaharan et al. [7] discutem os desafios ao se implementar detecção de intrusão de rede em Big Data, entre eles: o elevado custo computacional para se processar grandes volumes de dados e a falta de versatilidade que o uso de modelos de aprendizado supervisionado pode trazer. Com o intuito de endereçar o primeiro problema, Gupta et al. [8] propõem um *framework* de Big Data para detecção rápida de anomalias em redes, desenvolvido em Apache Spark, o que diminuiu significativamente o tempo de processamento; porém, os métodos adotados ainda utilizam aprendizado supervisionado.

Diante dos trabalhos anteriores, percebe-se a ausência de uma arquitetura que aborde, simultaneamente, o processamento em tempo ágil de volumes massivos de informação com uma modelagem da detecção de intrusão mais flexível, isto é, que lide com dados não rotulados. Portanto, este trabalho propõe uma arquitetura em Big Data capaz de analisar anomalias em tráfego de rede utilizando aprendizado não supervisionado em *near real time* - NRT[1]. A proposta consiste do uso do *framework* Apache Spark em conjunto com a técnica de clustering K-Prototypes, que também permite o uso de dados categóricos. A validação experimental da metodologia é feita em duas etapas: primeiramente com um conjunto de dados pré-rotulados, a fim de testar a eficácia da clusterização em tráfego de rede, e em seguida com a simulação de uma abordagem *online*, com tráfego real colhido de uma *Honeynet*, para validar a comunicação dos componentes e acurácia dos modelos gerados.

O restante do trabalho está assim organizado: a Seção II provê conceitos sobre componentes de sistemas de detecção de intrusão em Big Data, a Seção III descreve a arquitetura proposta, a Seção IV discute os resultados obtidos, e a Seção V tece as considerações finais.

Mateus Rocha e Daniel G. Silva, Departamento de Engenharia Elétrica, Universidade de Brasília, DF, e-mails: mateus.rocha@redes.unb.br, danielgs@ene.unb.br.

[1]Cenário onde a velocidade de processamento é importante, porém a aplicação é capaz de tolerar pequenos atrasos, na casa dos minutos [9].

II. Componentes de um Sistema de Detecção de Intrusão em Big Data

Os sistemas de detecção de intrusão em rede podem ser classificados como [10]: (*i*) sistema de detecção por assinatura, em que padrões identificadores (assinaturas) são procurados dentro dos dados, o que requer um banco de dados histórico sobre ataques, para assim distinguir o comportamento normal do suspeito; (*ii*) sistema de detecção por anomalia, em que se constrói um modelo que represente o comportamento normal do ambiente em questão, tal que qualquer atividade "diferente" o suficiente deste modelo é considerada suspeita.

Os sistemas de detecção de intrusão baseados em anomalias são os principais alvos de pesquisa, dada a contínua dinâmica de novas ameaças e ataques e sua maior capacidade em identificar comportamentos suspeitos, mesmo que desconhecidos. Ainda conforme [10], a detecção por anomalia pode ser baseada em aprendizado de máquina: a partir de um conjunto de dados (*dataset*) ilustrativo do comportamento da rede, cria-se um modelo capaz de categorizar as informações. Conseguir um bom *dataset* pode ser árduo e consumir muitos recursos, por outro lado, é benéfica a possibilidade de aprendizado contínuo (*online*), isto é, poder reajustar o modelo com novos dados para melhorar o seu desempenho.

Em seu artigo sobre aprendizado de máquina em sistemas de detecção de intrusão de rede, Tsai et al. [11] indicam que, com base em levantamento de trabalhos até 2010, apesar da grande variedade de técnicas, nenhum modelo obteve superioridade absoluta. Ademais, uma das maiores preocupações ao se gerar modelos de aprendizado de máquina é o seu custo computacional. Técnicas como *deep learning* podem não obter o desempenho necessário para rodar em aplicações do tipo *near real time*. Por outro lado, conforme o estudo feito por [12], utilizar o algoritmo não-supervisionado K-Means é vantajoso para analisar tráfego de rede rapidamente, dado o seu custo computacional relativamente baixo.

Com base nestas considerações e para se atingir os objetivos do trabalho — i.e. construir uma arquitetura de detecção de intrusão via tráfego de rede que rode em um cluster de *Big Data*, capaz de de processar elevados volumes de dados de maneira eficiente, distribuída, e o mais próximo do tempo real —, primeiramente, é necessário escolher um *framework* capaz de processamento distribuído; não obstante, deve-se associá-lo a uma metodologia baseada em aprendizado de máquina, responsável pela detecção de tráfego anômalo.

Com respeito ao primeiro aspecto, adota-se aqui o *framework* **Apache Spark**. Utilizado para processamento e análise de grandes volumes de informação [13], a ferramenta opera de maneira distribuída e pode processar paralelamente dados em memória primária, o que faz com que, de maneira geral, aplicações sejam executadas mais rapidamente do que na abordagem concorrente *MapReduce* [14]. A realização de operações em memória RAM é aspecto crucial no emprego deste *framework* dentro da arquitetura proposta, porque reduz o tempo para os cálculos do modelo de clusterização do tráfego de rede, possibilitando assim a sua análise em *near real time* [15].

Quanto ao segundo aspecto, a análise via aprendizado de máquina do tráfego de rede requer representá-lo por meio de atributos dos pacotes trafegados. Conforme [3], alguns dos principais atributos a serem analisados são: endereço IP, número de porta, *payload* e fluxo. Apesar de serem dados, em maior parte, numéricos, ainda há aqueles que são *strings*, logo se encaixam no conceito de atributos categóricos. Essa combinação de *features* numéricas e categóricas torna inadequado o uso do algoritmo K-Means original, pois este trabalha com a distância euclidiana, ideal para atributos numéricos.

O método **K-Prototypes** é uma variação do algoritmo K-Means que pode processar atributos tanto numéricos como categóricos, mantendo a rapidez e simplicidade do K-Means tradicional [16]. Sua métrica de distância entre os pontos dos *clusters* é dada pela fórmula $Dist_{total} = dist_{num} + \gamma.dist_{cat}$, em que γ determina o grau de importância das medidas categóricas em relação às numéricas e, por padrão, é calculado de acordo com o *dataset* de treinamento [17]. A função de distância entre os pontos categóricos é a *matching dissimilarity function* [18], onde o mesmo peso é atribuído para a presença ou ausência de caracteres. Para a implementação deste algoritmo, utilizou-se a biblioteca Python *kmodes* [19].

III. Proposta de Arquitetura

A arquitetura proposta neste trabalho visa detectar anomalias no tráfego de rede utilizando a técnica de clusterização K-Prototypes, conforme descrito na Seção II, ao mesmo tempo que busca utilizar ao máximo os recursos de paralelismo e resiliência providos pelo *framework* Apache Spark. Os componentes escolhidos buscam torná-la a mais modular possível. Dessa forma, espera-se obter uma solução versátil, permitindo que componentes sejam trocados ou reconfigurados de maneira dinâmica, conforme a necessidade do cenário.

A. Produtores e serviço de mensageria

O produtor (*producer*) é o componente responsável por capturar informações de tráfego e enviá-los ao cluster Big Data para que sejam processados. Essencialmente, há dois tipos de produtores: (*i*) do tipo *offline*, que visa validar e processar *datasets* já capturados e armazenados previamente, em que os principais tipos de arquivos consumidos por estes são CSV (*Comma Separated Values*) e capturas de rede, principalmente do tipo PCAP (*Packet Capture*); (*ii*) do tipo *online*, que captura e processa tráfego à medida em que são recebidos na máquina sob monitoramento, em termos concretos, a captura dos pacotes ocorre através do componente *tshark* [20].

Os dois tipos de *producers* enviam suas mensagens a um serviço de mensageria, para que fiquem disponíveis à aplicação Spark responsável por realizar a clusterização. Utiliza-se este tipo de serviço no lugar de uma arquitetura cliente-servidor porque se simplifica a implementação do consumidor de mensagens e se elimina a necessidade de monitoramento contínuo de alguma porta específica do sistema e de se alocar recursos computacionais para cada nova requisição que chega.

Neste sentido, adotou-se o serviço Apache Kafka devido à sua alta compatibilidade com serviços de Big Data [21]. A ferramenta age como *buffer* entre produtor e consumidor, aceitando mensagens a qualquer momento e entregando-nas aos consumidores apenas quando estão prontos [22].

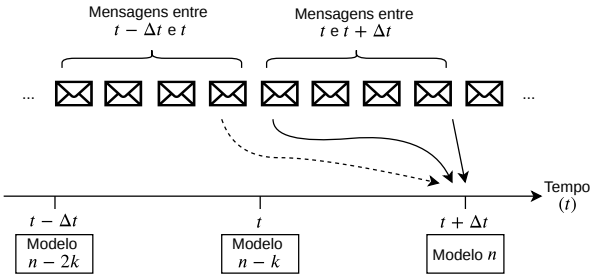

Fig. 1. Ilustração da periodicidade de treinamento do modelo de clusterização.

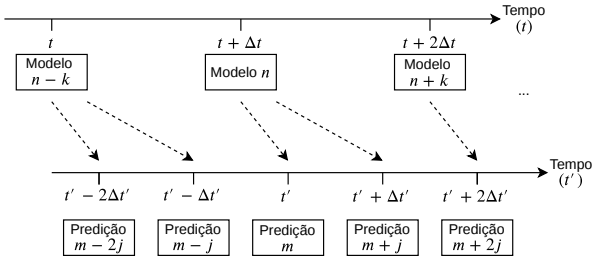

Fig. 2. Ilustração do processo contínuo de detecção de anomalias.

B. Treinador

A partir do momento em que as mensagens encontram-se disponíveis no serviço Kafka, elas podem ser consumidas pelos consumidores (*consumers*). Estes são programados para executar em intervalos de tempo fixos e, a cada nova execução, as mensagens mais recentes são lidas do serviço de mensageria e utilizadas para gerar um novo modelo de *clustering*, com base no algoritmo K-Prototypes. Dessa forma, os modelos utilizados para inferência são continuamente atualizados de acordo com a variação do fluxo de rede.

A Fig. 1 ilustra o processo de treinamento ao longo do tempo. É possível ver que um determinado modelo n, treinado em $t+\Delta t$, utiliza todas as mensagens recebidas desde o último modelo gerado (em t) em conjunto a algumas mensagens da janela de tempo anterior (entre $t-\Delta t$ e t), com o objetivo de se manter uma memória do aprendizado passado.

As informações extraídas do serviço Kafka são transformadas em um objeto RDD onde cada mensagem trocada entre cliente e servidor é representada por uma linha contendo diversos campos, de tal forma que a quantidade de linhas equivale ao total de pacotes de rede trafegados no período.

Após a clusterização através do método K-Prototypes, um objeto Python é gerado contendo o melhor modelo dentre as iterações realizadas durante o treinamento[2]. Cada iteração corresponde a uma inicialização distinta dos centróides e a escolha do melhor modelo é feita ao selecionar aquele de menor soma das distâncias intra-clusters. Esse objeto então é salvo no *Hadoop Distributed File System* (HDFS) em formato *pickle* para que possa ser restaurado posteriormente.

C. Predição

A predição é feita por outro processo, também escrito para rodar em Apache Spark de maneira periódica e independente. Dessa forma, o intervalo de tempo para re-execução deste módulo é diferente daquele utilizado pelo treinador. Conforme pode ser visto na Fig. 2, a predição acontece de maneira mais frequente que o treinamento de novos modelos, visto que a detecção requer resultados mais rapidamente e com tempos de processamento menores. Para cada nova execução do preditor, busca-se o modelo mais recentemente treinado e coletam-se todas as novas mensagens que chegaram ao serviço Apache Kafka desde a execução anterior.

[2]Adota-se o número padrão indicado pela ferramenta, que é de 100 [17].

Após a coleta, o sistema carrega do HDFS o último modelo treinado e classifica os novos pacotes recebidos com base em uma regra de decisão configurável: se o pacote não pertence a um dos clusters identificados na etapa de treinamento, isto é, caso seu vetor de atributos possua distância ao centróide de qualquer cluster maior que um limiar de decisão pré-configurado, ele é classificado como anômalo e incluído no relatório criado pela ferramenta após cada ciclo.

IV. Validação experimental da arquitetura

Para a validação da arquitetura, utiliza-se duas abordagens: primeiramente, utiliza-se um *dataset* rotulado para se mensurar a acurácia do modelo treinado de forma não-supervisionada em relação às classes reais; posteriormente, após validar o devido desempenho, é feita a simulação do cenário *online*, onde o algoritmo de clusterização é executado sobre dados gerados a partir de um servidor real e, em sequência, o sistema é submetido a ataques de rede, para testar a eficácia da detecção do tráfego malicioso.

A. Análise com dados pré-rotulados

O *dataset* a ser analisado é o *Intrusion Detection* [23]. Ele contém exemplos de atributos descritores de tráfego de rede malicioso e de tráfego normal, com a respectiva classificação em: dos, probe, r2l, u2r (ataques) e normal. O conjunto de treinamento possui aproximadamente 126 mil exemplos e o conjunto de teste possui 10 mil exemplos.

Primeiramente, uma etapa de seleção de atributos é necessária; após uma avaliação qualitativa, selecionou-se apenas os atributos majoritariamente referentes a tráfego de rede, vide Apêndice A. Em seguida, filtrou-se no conjunto de treinamento apenas os 68 mil exemplos (aproximadamente) da classe "normal", pois o objetivo da ferramenta é criar um modelo para a situação de normalidade de tráfego, de modo que qualquer dado futuro que caracterize um comportamento que não se encaixe dentro desta modelagem e, portanto, que possa representar tráfego malicioso, seja capaz de se identificar.

Realizada a etapa de pré-processamento, utilizou-se o "método do cotovelo" para cálculo do número adequado de clusters, chegando a $K = 4$. Quanto ao limiar de decisão, isto é, a distância máxima até o centróide de um cluster para que se considere que um ponto pertença àquele grupo, utilizou-se o dobro da distância intracluster, isto é, a média da distância entre os pontos de um cluster e seu respectivo centróide (2σ).

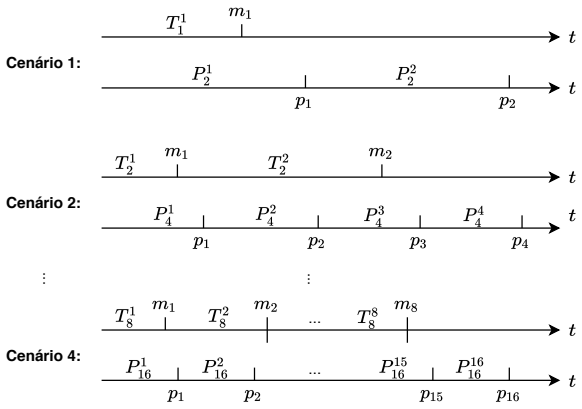

Fig. 3. Descrição dos 4 cenários de simulação, para os dados pré-rotulados.

TABELA I
TEMPO DE EXECUÇÃO DOS PROCESSOS DE TREINAMENTO E PREDIÇÃO NOS CENÁRIOS TESTADOS.

	Cenário	Qtd de ciclos	Fração do dataset por ciclo	Tempo médio	Msgs por ciclo
	1	1	T/1	1600 s	67343
	2	2	T/2	725 s	33671
	3	4	T/4	335 s	16835
Treino	4	8	T/8	170 s	8417
	1	2	P/2	0,598 s	5000
	2	4	P/4	0,419 s	2500
	3	8	P/8	0,267 s	1250
Predição	4	16	P/16	0,140 s	625

Para uma simulação mais próxima do real, um script Python foi desenvolvido para dividir os datasets em n partes e submeter, ao poucos, as mensagens ao serviço Kafka de modo que o módulo treinador seja capaz de gerar vários modelos ao longo do tempo, com o preditor capaz de classificar novos blocos de mensagens por meio do modelo mais recentemente gerado. A Fig. 3 ilustra os cenários, T_m^n indicam as frações do dataset de treinamento, P_j^k são as frações do dataset de predição, m_x são os modelos gerados ao longo do tempo e p_y retratam as predições realizadas nas simulações. Ao todo, foram testados quatro cenários com um crescente particionamento dos datasets, de modo que cada ciclo de operação do Spark tivesse que processar uma quantidade menor de mensagens. Durante os testes, manteve-se a proporção de dois ciclos de predição para cada novo modelo treinado.

Primeiramente temos, na Tabela I, os resultados em termos de custo computacional dos diferentes cenários. Como pode ser visto, o tempo de predição é bem menor que os tempos de treinamento, logo é vantajoso para o sistema que ambas as entidades estejam em módulos separados e independentes. Também é possível perceber que dividir a quantidade de mensagens pela metade faz com que o treinamento tenha seu tempo reduzido aproximadamente na mesma proporção. Porém, é importante ressaltar que o intervalo entre retreinos não seja muito curto para garantir que uma quantidade suficientemente grande de mensagens seja utilizada para gerar modelos de qualidade. Enquanto isso, diminuir o intervalo de predição mostrou-se vantajoso para a arquitetura, pois o sistema é capaz de retornar os resultados mais rapidamente.

Por sua vez, a Tabela II contém os resultados em termos de desempenho na detecção, para os quatro cenários. No comparativo entre os quatro cenários, é possível perceber que não há degradação nos resultados obtidos ao se aumentar a frequência da inferência.

TABELA II
RESULTADO DOS CENÁRIOS TESTADOS, PARA O DATASET ROTULADO.

Cenário	Acurária média	Precisão média	Recall médio	Especificidade média	F1 Score médio
1	79,84%	87,91%	73,54%	87,62%	80,32%
2	80,34%	88,58%	74,78%	87,74%	82,46%
3	80,52%	88,96%	74,66%	87,81%	81,16%
4	81,09%	88,89%	76,15%	87,55%	82,01%

B. Análise em regime online

Para validar o funcionamento *online* da arquitetura, analisou-se o tráfego de rede do servidor de *proxy*-reverso do laboratório pertencente ao departamento de Engenharia Elétrica da universidade. Com base na arquitetura de rede do laboratório, o *proxy*-reverso está situado dentro de uma rede protegida por Firewall. Este, por sua vez, possui regras para descartar pacotes suspeitos (destinados a portas específicas que não englobam o protocolo HTTP e seus derivados), então se assume que o tráfego coletado deste servidor pode ser classificado como não-malicioso.

Para simular ataques de rede reais ao sistema, utilizou-se o tráfego HTTP derivado da *Honeynet* montada também no mesmo departamento, resultado dos trabalhos [24] e [25]. Um *script* em Python é usado para ler os PCAPs armazenados, extrair apenas os atributos dos pacotes e submetê-los ao serviço Kafka, de acordo com seus *timestamps* originais. Assim, foi possível submeter a arquitetura a ataques sem de fato comprometer o *cluster* Big Data.

Conforme [3], os endereços IP dos pacotes e seus números de porta são boas *features* para se utilizar na classificação de tráfego de rede. Estes e os demais atributos escolhidos estão no Apêndice B. Novamente aplicou-se a "regra do cotovelo" para definição do número de clusters, chegando a $K = 3$.

Com as etapas preliminares realizadas, passou-se à execução do processo treinador, de forma a gerar novos modelos periodicamente. Com base na quantidade de tráfego recebida pelo servidor, escolheu-se o período de 1 hora para retreino, visto que em média aproximadamente 6000 pacotes eram trocados por ele neste intervalo de tempo. Com base no experimento anterior, essa quantidade de pacotes é capaz de gerar um novo modelo em um intervalo relativamente curto ($< 170s$).

Então, com auxílio da ferramenta *tshark* em conjunto com o *script* Python previamente descrito, o tráfego malicioso é continuamente submetido ao serviço Kafka, respeitando os seus timestamps originais, alternado com o tráfego legítimo, para o processamento pelo módulo de predição. Este é configurado para rodar no dobro da frequência do treinador, ou seja, a cada 30 minutos. Os resultados são mostrados na Tabela III.

Conforme pode ser visto na tabela, o tempo médio necessário para fazer a predição das mensagens recebidas em cada ciclo do preditor foi de 1,53 segundos. Dessa forma, é possível configurar esse módulo para rodar em intervalos menores que 30 minutos, por exemplo, de 1 em 1 minuto.

TABELA III
RESULTADOS DA PREDIÇÃO NO TRÁFEGO DO PROXY REVERSO.

T_{treino} médio	$T_{predicao}$ médio	Acurácia média	Precisão média	Recall médio	Specifity média	F1 Score médio
71 s	1,5289 s	89,69%	73,87%	87,33%	90,30%	77,54%

É possível perceber também que as métricas de classificação foram condizentes com aquelas encontradas no experimento anterior. Conforme esperado, este experimento levou menos tempo em média para gerar os modelos, pois cada ciclo possuía menos mensagens que o Cenário 4 testado na Seção IV-A.

V. CONCLUSÕES

Este trabalho propõe uma arquitetura em Big Data de detecção em tempo quase-real (NRT) de anomalias em tráfego de rede via clusterização. O sistema se mantém atualizado ao, periodicamente, retreinar o modelo preditivo com novos dados, permitindo um acompanhamento da dinâmica de tráfego. Ademais, as tarefas de treinamento e subsequente detecção/predição estão implementadas em módulos separados, o que permite ao treinador utilizar mais dados (e consequentemente demorar mais) para gerar modelos mais robustos e ao preditor fazer análises mais frequentemente.

A utilização do *framework Spark* permitiu a captura e processamento do tráfego de rede de maneira transparente, com pleno uso dos recursos de paralelismo. Todos os componentes criados foram modularmente desenvolvidos para permitir a manutenção e evolução da ferramenta por partes. A técnica de clusterização K-Prototypes apresentou resultados eficientes no processamento de altas quantidades de tráfego de rede com baixa latência, com o benefício da incorporação de *features* categóricas. Por fim, obteve-se resultados satisfatórios para a detecção de anomalias nos cenários de testes realizados.

Como trabalho futuro, espera-se analisar a arquitetura com o modelo de detecção baseado em outros protocolos da camada de aplicação como o DNS, o Telnet e o SSH; comparar os resultados obtidos com outros trabalhos relacionados; testar os modelos gerados com o auxílio de ferramentas que simulam ataques de rede reais, tais como o HPing; e, por fim, adicionar mecanismos de visualização para facilitar a análise dos resultados obtidos pelo usuário do sistema.

REFERÊNCIAS

[1] Symantec, "2019 internet security threat report," 2019.

[2] E. Bloedorn, A. D. Christiansen, W. Hill, C. Skorupka, L. M. Talbot, and J. Tivel, "Data mining for network intrusion detection: How to get started," tech. rep., Citeseer, 2001.

[3] H. G. Kayacik, A. N. Zincir-Heywood, and M. I. Heywood, "Selecting features for intrusion detection: A feature relevance analysis on kdd 99 intrusion detection datasets," in *Proceedings of the third annual conference on privacy, security and trust*, vol. 94, pp. 1723–1722, 2005.

[4] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, 2016.

[5] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Clustering-based network intrusion detection," *International Journal of reliability, Quality and safety Engineering*, vol. 14, no. 02, pp. 169–187, 2007.

[6] L. Wang and R. Jones, "Big data analytics for network intrusion detection: A survey," *International Journal of Networks and Communications*, vol. 7, no. 1, pp. 24–31, 2017.

[7] S. Suthaharan, "Big data classification: Problems and challenges in network intrusion prediction with machine learning," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 4, pp. 70–73, 2014.

[8] G. P. Gupta and M. Kulariya, "A framework for fast and efficient cyber security network intrusion detection using apache spark," *Procedia Computer Science*, vol. 93, pp. 824–831, 2016.

[9] C. Wilson, "The difference between real time, near-real time, and batch processing in big data," 2015.

[10] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *computers & security*, vol. 28, no. 1-2, pp. 18–28, 2009.

[11] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *expert systems with applications*, vol. 36, no. 10, pp. 11994–12000, 2009.

[12] L. Portnoy, *Intrusion detection with unlabeled data using clustering*. PhD thesis, Columbia University, 2000.

[13] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[14] Y. Samadi, M. Zbakh, and C. Tadonki, "Performance comparison between hadoop and spark frameworks using hibench benchmarks," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 12, p. e4367, 2018.

[15] E. F. Z. Santana, "Introdução ao apache spark," 2016.

[16] Z. Huang, "Clustering large data sets with mixed numeric and categorical values," in *Proceedings of the 1st pacific-asia conference on knowledge discovery and data mining,(PAKDD)*, pp. 21–34, Singapore, 1997.

[17] N. de Vos, "kmodes 0.10.1," 2019.

[18] R. R. Sokal, "A statistical method for evaluating systematic relationships," *Univ. Kansas, Sci. Bull.*, vol. 38, pp. 1409–1438, 1958.

[19] N. J. de Vos, "Kmodes categorical clustering library." https://github.com/nicodv/kmodes, 2015–2020.

[20] U. Lamping and E. Warnicke, "Wireshark user's guide," *Interface*, vol. 4, no. 6, 2004.

[21] V. Starostenkov and K. Grigorchuk, "Hadoop distributions: Evaluating cloudera, hortonworks, and mapr in micro-benchmarks and real-world applications," *Altoros Systems Inc*, 2013.

[22] K. M. M. Thein, "Apache kafka: Next generation distributed messaging system," *International Journal of Scientific Engineering and Technology Research*, vol. 3, no. 47, pp. 9478–9483, 2014.

[23] Kaggle, "Intrusion detection," 2017.

[24] G. A. de Oliveira Júnior, R. T. de Sousa Júnior, and D. F. Tenório, "Desenvolvimento de um ambiente honeynet virtual para aplicação governamental," 2015.

[25] G. A. d. Oliveira Júnior, "Honeyselk: um ambiente para pesquisa e visualização de ataques cibernéticos em tempo real," 2016.

APÊNDICE

A. Campos escolhidos no dataset Intrusion Detection

duration, protocol_type, service, flag, src_bytes, dst_bytes, count, srv_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate, dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate.

B. Campos escolhidos na análise online

ip.src, ip.proto, ip.ttl, ip.hdr_len, ip.len, http.connection, http.content_encoding, http.content_length, http.content_length_header, http.host, http.referer, http.request, http.request.full_uri, http.request.method, http.request.uri, http.request.version, http.request_in, http.request_number, http.response, http.response.code, http.response.phrase, http.response_number, http.server, http.time, http.user_agent.