A Monitoring and Threat Detection System Using Stream Processing as a Virtual Function for Big Data

Martin Andreoni Lopez[1,2], Otto Carlos M. B. Duarte[1] and Guy Pujolle[2]

[1]GTA / PEE-COPPE / UFRJ – Brazil [2]LIP6/CNRS Sorbonne Université – France

Abstract. *The late detection of security threats causes a significant increase in the risk of irreparable damages, disabling any defense attempt. As a consequence, fast real-time threat detection is mandatory for security guarantees. In addition, Network Function Virtualization (NFV) provides new opportunities for efficient and low-cost security solutions. We propose a fast and efficient threat detection system based on stream processing and machine learning algorithms. The main contributions of this work are i) a novel monitoring threat detection system based on stream processing; ii) two datasets, first a dataset of synthetic security data containing both legitimate and malicious traffic, and the second, a week of real traffic of a telecommunications operator in Rio de Janeiro, Brazil; iii) a data pre-processing algorithm, a normalizing algorithm and an algorithm for fast feature selection based on the correlation between variables; iv) a virtualized network function in an open-source platform for providing a real-time threat detection service; v) near-optimal placement of sensors through a proposed heuristic for strategically positioning sensors in the network infrastructure, with a minimum number of sensors; and, finally, vi) a greedy algorithm that allocates on demand a sequence of virtual network functions.*

1. Introduction

Traffic monitoring is critical to maintaining the stability, reliability, and security of computer networks [Hu et al. 2015]. Network monitoring extends from a simple collection of statistics to a complex analysis of upper-layer traffic for tuning network performance and debugging protocols. Current network monitoring tools, such as NetFlow, SNMP, Bro, or Snort, do not meet the current speed and management needs of large network domains. In addition, many of these tools generate a huge number of files that require processing of other types of tools to extract knowledge from the collected data. Thus, current security systems such as Security Information and Event Management (SIEM) do not perform satisfactorily since, while 82% of security threats occur in minutes, an intrusion can take up to 8 months to be detected. It is essential that the detection time is the least possible so that the intrusion prevention can be effective [Wu et al. 2014].

Security attacks have been improving and simple analysis and filtering of packets by the IP header and TCP port are no longer effective. The attacking traffic tries to hide itself from the security tools by forging the source IP and dynamically changing the port. In this context, a promising alternative for classifying traffics and detecting threats is the use of Machine Learning (ML) techniques. These techniques are suitable for large masses of data because, with more samples to train the classifier, the methods tend to be more accurate [Mayhew et al. 2015]. However, with large amounts of data, machine learning methods have high latency due to the consumption of computing time. This high latency is a disadvantage for machine learning methods that must analyze data and detect threats as quickly as possible. In this context, real-time stream processing allows the immediate analysis of different types of data and consequently benefits traffic monitoring for security threat detection. Open source distributed processing

platforms have recently been proposed to process large masses of data in low latency flows. To do so, these platforms are the basis for developing custom applications for each case.

This work proposes A s**CA**lable **TRA**ffic **C**lassifier and **A**nalyzer (CATRACA) tool. CATRACA uses Network Function Virtualization (NFV) technology and its infrastructure to combine virtualization, cloud computing, and distributed stream processing to monitor network traffic and detect threats. The goal is to provide an accurate, scalable and real-time threat detection tool capable of meeting peaks of use, providing a high Quality of Service. Traffic monitoring and threat detection as a virtualized network function have two main advantages: the ability to self-adapt to different traffic volumes and the flexibility of installation and migration of sensors in the network to reduce the latency in monitoring [Andreoni Lopez et al. 2016]. Thus, the tool analyzes big data, the Machine Learning techniques classify the traffic into normal or threat, and, finally, the knowledge extracted from flows is presented in a user interface. [1]

2. Objectives and Contributions

The goal of this work is to present the research and the obtained results achieved during the thesis process. The research topics assessed are Stream Processing, Real-Time Threat Detection System, Dataset and Feature Selection, Virtual Network Function, and Virtual Network Function Chaining.

Stream Processing. We analyze and compare two native distributed real-time and stream-processing systems, Apache Storm [Toshniwal et al. 2014] and Apache Flink [Carbone et al. 2015], and one micro-batch system, the Apache Spark Streaming [Franklin 2013]. The architecture of each analyzed system is discussed in depth and a conceptual comparison is presented showing the differences between these open-source platforms. Furthermore, we evaluate the data processing performance and the behavior of systems when a worker node fails.

Dataset and Data Preprocessing. We created two datasets, first a synthetic security dataset to perform traffic classification and the second one is a real traffic from a network operator in Rio de Janeiro, Brazil. Furthermore, we present a fast preprocessing method for network traffic classification based on feature correlation and feature normalization. Our proposed method couples a normalization and a feature selection algorithms. We evaluate the proposed algorithms against three different datasets for eight different machine learning classification algorithms. Our proposed normalization algorithm reduces the classification error rate when compared with traditional methods. Our Feature Selection algorithm chooses an optimized subset of features improving accuracy by more than 11% within a 100-fold reduction in processing time when compared to traditional feature selection and feature reduction algorithms.

Real-Time Threat Detection System. We propose and implement an accurate real-time threat detection system, the CATRACA[2] tool [Andreoni Lopez et al. 2017b]. The integrated system allows big data analytic in a stream processing manner. The proposed system uses machine learning for both attack classification and threat detection. Moreover, the system has a friendly graphical interface that provides a real-time visualization of the parameters and the attacks that occur in the network.

Virtual Network Function. We evaluate CATRACA as a Virtual Network Function (VNF) in the Open Source Platform for Network Functions Virtualization (OPNFV) that pro-

[1]The tool, as well as its documentation and complementary information can be accessed at http://gta.ufrj.br/catraca

[2]documentation available at `http://catraca.gta.ufrj.br/` Accessed February 2019.

vides an accurate real-time threat detection service. The service provided is able to scale the number of processing cores by adding virtual machines to the processing cluster that executes the detection in a parallel-distributed way, processing up to 15 Million samples per minute. Besides, the Network Virtualization Platform enables the easy deployment of traffic capture sensor elements that can be placed and moved to several points in the network, offering customization and adaptability to network monitoring. The results show the potential for scalability, as we increase the number of processing cores in the distributed cluster. Another important feature of our proposal is the migration of processing machines. The experiments show that our system can migrate the processing elements without stopping the threat detection. The live migration enables the organization of the physical machines in the processing cluster, which results in several advantages, such as shutting down machines for maintenance or for reduction of energy consumption or allocating resources in a smart way to meet the demand.

Virtual Network Function Chaining. We propose a scheme for placing and chaining Virtual Network Functions over a network according to four different heuristics. The first heuristic places the VNF nodes into physical nodes that introduce the minimum delay between the traffic source and destination. The second heuristic searches for the best placement of VNF nodes considering the nodes that have the biggest amount of available resources and, thus, places the VNF over the most available node. This approach increases the number of accepted requests of VNFs in a network. The third heuristic places the VNF nodes according to the betweenness-centrality of the topology nodes. In the betweenness-centrality approach, the requests are primarily responded by allocating the most central nodes on the topology, which reduces the introduced delay. However, as the resources of the most central nodes are used, the following requests are allocated into peripheral network nodes, introducing a greater delay on the VNF chaining. The fourth heuristic weights the available resources and the introduced delay for each physical node. Then, it allocates the VNFs on the nodes that present the greatest probability of supplying enough resources and the lowest delay. We deploy a greedy algorithm for all four approaches and we simulate the allocation of VNFs over a real network topology.

3. Related Work

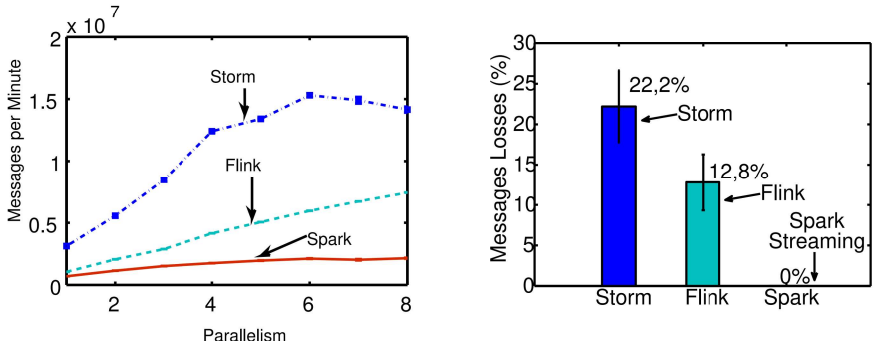
There are some proposals that use the Storm stream processing tool to perform real-time anomaly detection. Du *et al.* use the Flume and Storm tool to traffic monitoring to detect anomalies. The proposal is to make the detection through the k-NN algorithm [Du et al. 2014]. The article presents some performance results, but it lacks evaluation of the accuracy of detection and the tool does not receive data from multiple sources. The work of Zhao *et al.* uses the Kafka and Storm, as well as the previous work, for the detection of network anomalies [Zhao et al. 2015], characterizing flows in the NetFlow format. He *et al.* propose a combination of the distributed processing platforms Hadoop and Storm, in real time, for the detection of anomalies. In this proposal, a variant of the k-NN algorithm is used as the anomaly detection algorithm [He et al. 2015]. The results show a good performance in real time, however without using any process of reaction and prevention of the threats. Mylavarapu et al propose to use Storm as a flow processing platform in the [Mylavarapu et al. 2015] intrusion detection. The Stream4Flow proposal uses Apache Spark with the ElasticStack stack to do network monitoring [Jirsik et al. 2017]. The prototype serves as a visualization of network parameters. Stream4Flow, however, has no intelligence to perform anomaly detection. Dos Santos *et al.* Use a combination of Snort IDS and OpenFlow to create Of-IDPS.

The OpenSOC project: The Open Security Operations Center [Santos 2015] is a collaborative development project that integrates several open source software aimed at an extensible and scalable security analysis tool. Thus, OpenSOC is an analytical security framework for

monitoring large masses of data using distributed stream processing. OpenSOC was discontinued and gave rise to the Apache Metron project [Apache Software Foundation 2017] which is an evolution of OpenSOC and proposes a new architecture that aims to facilitate the addition of new data sources, and better exploit the parallelism of the Storm tool.

The proposed CATRACA tool, like Metron, was also inspired by OpenSOC and aims to monitor large volumes of data using flow processing. The CATRACA tool is implemented as a virtualized network role (VNF) in the Open Platform for Network Function Virtualization (OPNFV) environment. CATRACA focuses on real-time packet capture, feature selection, machine learning, and has a mechanism of action for immediate blocking of malicious flows. Thus, the CATRACA tool acts as a virtualized network intrusion detection and prevention function that reports flow summaries and can be linked to other network virtualized functions [Sanz et al. 2017] as defined in the network function chain patterns (Service Function Chaining - SFC) and network service headers (Network Service Header - NSH).

4. Obtained Results



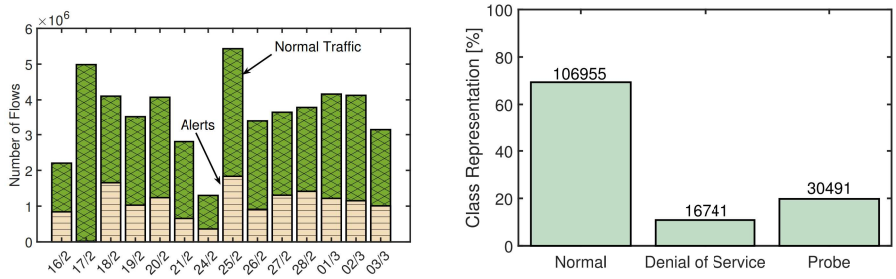
(a) Stream Processing Frameworks comparison.

(b) Message losses during node failure.

Figure 1: Throughput results of the platforms in terms of number of messages processed per minute in function of the task parallelism.

We evaluate the performance of stream processing platforms in terms of analyzed messages and node failure. The full content of the GTA/UFRJ dataset is injected into the system and then it is replicated as many times as necessary to create a large volume of data. The experiment calculates the rate of consumption and messages analyzed by each platform. Also, the parallelism parameter was varied, which represents the total number of cores available for the cluster to process samples in parallel. Figure 1a shows the results of the experiment where Apache Storm has the highest throughput. For a single core, unparalleled, Storm already shows better performance with a flow rate at least 50% higher when compared to Flink and Spark streaming. Flink has a linear growth, but with values always inferior to those of Apache Storm. The processing rate of Apache Spark streaming, when compared to Storm and Flink, is much lower and this is due to the use of a microbatch. Each microbatch is pooled prior to processing, generating a delay in each processed sample. Apache Storm behavior is linear up to four-core parallelism. Then, the processing rate grows until the parallelism of six, in which the system saturates. This behavior was also observed in Apache Spark streaming with the same parallelism of six cores. We also analyze the behavior of the platforms during a node failure. Figure 1b shows the comparison of lost messages, where Spark had no loss during the fault. The measure shows the percentage of lost messages by systems, calculated by the difference of messages sent by Apache Kafka and messages analyzed by the systems. Thus, Apache Flink has a smaller loss of messages during a fault with about a 12.8% compared to 22.2% in Storm.

We create two datasets to evaluate CATRACA system. Figure 2b show the relation of classes used in the artificial UFRJ/GTA dataset. The Normal class is around 70% of the dataset with 106.955 samples. The Denial of Service (DoS) class is 10% of the total dataset with 16.741 samples, and, finally, Probe class represents almost the 20% of the dataset with 30.491 samples. In the Network Operator (NetOp) dataset packets are first anonymized, then PPPoE encapsulation is removed. An Intrusion Detection System (IDS) is used to classify alerts, in parallel, packets are abstracted in 43 flow features. Finally, an application is used to match traffic flows with IDS alerts, generating a flow with 44 features corresponding to 1 if alert and 0 to normal traffic. On the other hand, Figure 2a shows the number of threats and normal flow in each day of the dataset in 2017. We can see that almost all days contains around 30% of alerts. Only day 17/2 contains a smaller number of alerts. The maximum alerts number was during the Saturday 25/2 with 1.8 Million alerts.



(a) NetOp Dataset alerts distribution. **(b) GTA/UFRJ classes distribution.**

Figure 2: a) Number of Alerts and Normal Traffic flows in Network Operator dataset. b) Classes Distribution in the Dataset. The main class is the Normal with almost 70% of the dataset, DoS is around 10% and Probe correspond to 20% of the Dataset.

A pre-processing algorithm is proposed to accelerate the training and classification time used in algorithms. Figure 3 shows a comparison of processing time of all implemented feature selection and dimensionality reduction methods. Our proposal is compared with a dimensionality reduction method, the Principal Component Analysis (PCA), a filter feature selection method Relief, a wrapper method, Sequential Feature Selection (SFS) and an embedded method Support Vector Machine Recursive Feature Elimination (SVM-RFE). All measures are in relative value. We can see that SFS show the worst performance. The SFS algorithm performs multiple iteration in order to minimize the mean square error (MSE). Consequently, all these iterations increase the processing time. Our proposal shows the best processing time together with PCA, because both implementations perform a matrix multiplication.

Figure 4a shows the training and classification times with no feature selection, while Figure 4b shows the training and classification times for the dataset with 90% of feature reduction. All the classifiers reduced their times. K-NN training time is reduced by 71%, while classification time is reduced by 84%. For Neural Networks reduced the training time by 25% and classification time is reduced in 0.02 seconds. Random Forest reduced their training time by 38% while their classification time remains the same. SVM with RBF kernel training time is reduced by 78% and training time is reduced by 54%. SVM with linear kernel received the biggest improvement. Training time was reduced by 88% while classification time was reduced by 81%. Gaussian Naive Bayes reduced their training time in 80% while classification time was reduced in 76%. Stochastic Gradient Descendant also shows a reduction of 61% in training and 66% for classification time. Finally, Decision Tree reduced training time by 79% and classification time got faster, being reduced by 28%. Thus, a feature reduction of 90% impacts directly in the training and classification time of the machine learning classifiers. Therefore, our Feature

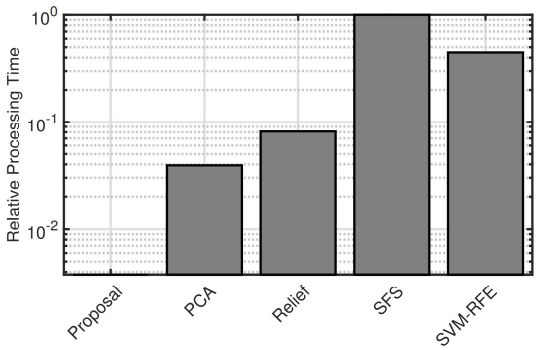
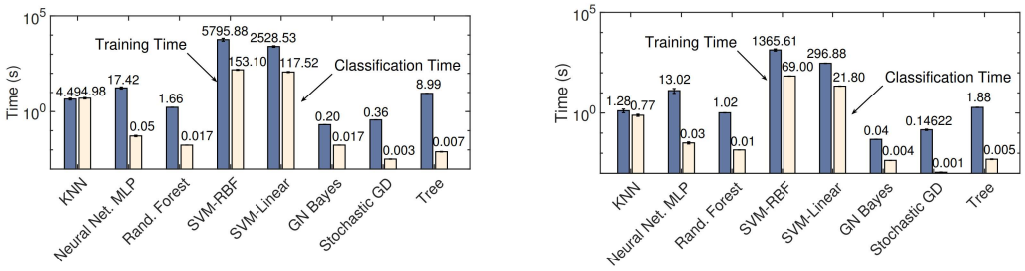


Figure 3: Performance of features selection algorithms. Performance of features selection algorithms according to processing time. The proposal and the PCA show the best processing time.



(a) No feature Selection. **(b) 90% Features Reduction.**

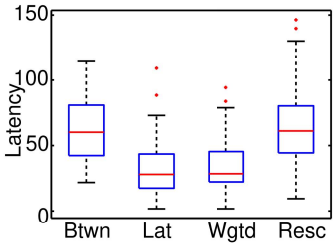
Figure 4: Our proposed feature selection method when applied to different classification algorithms. a) No feature selection applied. b) 90% Features Reduction. Results show training and classification time in NetOp Dataset.

Selection method improves training and classification times in all classifiers.

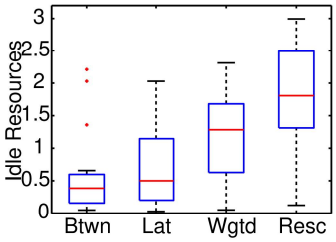
An infrastructure provider can allocate different Virtual Network Functions (VNFs) to supply a new service. The deployment and reconfiguration of several VNFs is known as Virtual Function Chaining. A request is a sorted list of VNFs that describe the order in which traffic has to be processed. Therefore, the allocation of the request on the network needs to consider the order of the VNFs as well as the source and the destination of the traffic handled by the set of VNFs in the request. We also study when allocating a VNF over a physical node, the physical node has to provide enough resources to answer the needs of all hosted VNFs. Our proposed scheme is composed of two main phases. First, we estimate the resources available on the physical nodes and the resources requested by the VNFs. The second phase is to run a greedy algorithm that takes as input the VNF requests as they arrive, and then it places each VNF on a physical node that has enough resources. Our greedy algorithm considers four different heuristics to place the VNFs on the network. CPU, memory, and network resources on physical and virtual nodes must be considered for VNF allocation. In order to summarize all resources into one single variable, we consider the `Volume` metric introduced by Wood *et al.* [Wood et al. 2009]. Thus, for each VNF the `volume` metric is the ratio of the resources on physical node that the VNF is requesting. The VNF `volume` ranges from 0 to 1, where 1 means that a VNF is requesting an entirely available physical node to be installed.

On the second phase, a greedy algorithm adopts one of the four heuristics, **minimum latency**, chooses the node that introduces a minimum delay to the path, in comparison to the previous selected nodes to host the other VNFs, or the source of the traffic; **maximum usage of resources**, selects the node with the biggest amount of available resources to host a VNF,

without considering the routing constrains between the already placed VNFs; **most central nodes**, designate to place the VNF into the most central node, i.e. the node that presents the greatest betweenness-centrality value, and has enough resources to host the VNF; **weighted latency and resource**, in which the probability of choosing each a node for hosting a VNF is weighted based on the latency that it introduces to the path and the available resources that it has. The weight of the node i is given by $w_i = \left(1 - \frac{lat_i}{max_{j \in N}(lat_j)}\right) \times \left(\frac{rec_i}{max_{j \in N}(rec_j)}\right)$, where lat_i stand for the latency introduced by node i, rec_i is the available resources in node i, and N is the set of all nodes in the network. The greedy algorithm searches for hosting VNFs on the nodes that have the biggest w_i value first.



(a) Dispersion of the latency distribution into the allocated VNFs.



(b) Dispersion of the remaining idle resource distribution after allocating all VNFs.

Figure 5: a) The minimum latency heuristic introduces the lower average delay on the packet-processing path. The maximum resource usage heuristic is the one that presents the greatest dispersion into the latency distribution thanks to ignoring the latency concerns when placing the VNFs. b) The Maximum Resource heuristic presents the most distributed remaining resources.

As shown in Figure 5a, the minimum latency heuristics introduces the lowest average delay on the packet-processing path. The maximum resource usage heuristic is the one that presents the greatest dispersion into the latency distribution thanks of ignoring the latency concerns when placing the VNFs. This result shows that the Latency heuristic reduces 52% of the average delay when compared with the betweeness-centrality heuristic. Moreover, the latency heuristic also achieves the greatest number of accepted VNF requests, which have the minimum latency, even when compared with maximum resource allocation that achieves to allocate more requests than all others. Figure 5b shows the remaining resources after all VNF allocation. Although the maximum resource heuristic instantiates more VNFs, it presents the biggest amount of idle resources. Nevertheless, it is also the most distributed idle resource pattern, which implies a load distribution between all physical nodes.

5. Final Considerations

The proposals discussed in this thesis have a contribution in the area of network security. The work proposed the CATRACA [Andreoni Lopez et al. 2017b][3], a virtual network function for threat detection. CATRACA combines machine learning, stream processing and network function virtualization. Two datasets were created [Andreoni Lopez et al. 2017c][4] to evaluate the threat detection, and an algorithm for preprocessing stream data [Andreoni Lopez et al. 2018b]. We finally evaluated the performance of CATRACA [Andreoni Lopez et al. 2018a] as a virtual network function and propose an algorithm for virtual function chaining [Andreoni Lopez et al. 2017a].

[3]Best demo in Salão de Ferramentas do SBSeg'17.

[4]Best paper award CSNet'17.

References

Andreoni Lopez, M., Lobato, A. G. P., Duarte, O. C. M. B. e Pujolle, G. (2018a). An evaluation of a virtual network function for real-time threat detection using stream processing. Em *IEEE Fourth International Conference on Mobile and Secure Services (MobiSecServ)*, páginas 1–5.

Andreoni Lopez, M., Mattos, D. e Duarte, O. (2017a). Evaluating allocation heuristics for an efficient virtual Network Function chaining. Em *2016 7th International Conference on the Network of the Future, NOF 2016*.

Andreoni Lopez, M., Mattos, D. M. F. e Duarte, O. C. M. B. (2016). An elastic intrusion detection system for software networks. *Annales des Telecommunications/Annals of Telecommunications*, 71(11-12):595–605.

Andreoni Lopez, M., Mattos, D. M. F., Duarte, O. C. M. B. e Pujolle, G. (2018b). A fast unsupervised preprocessing method for network monitoring. *Annals of Telecommunications*.

Andreoni Lopez, M., Sanz, I. J., Ferrazani Mattos, D. M., Duarte, O. C. M. B. e Pujolle, G. (2017b). CATRACA: uma Ferramenta para Classificação e Análise Tráfego Escalável Baseada em Processamento por Fluxo. Em *Salão de Ferramentas do XVII SBSeg'2017*, páginas 788–795.

Andreoni Lopez, M., Silva, S. R., Alvarenga, I. D., Rebello, G. A. F., Sanz, J. I., Lobato, A. G. P., Mattos, D. M. F., Duarte, O. C. M. B. e Pujolle, G. (2017c). Collecting and Characterizing a Real Broadband Access Network Traffic Dataset. Em *IEEE/IFIP 1st Cyber Security in Networking Conference (CSNet'17)*, Rio de Janeiro, Brazil.

Apache Software Foundation (2017). Apache Metron. `https://cwiki.apache.org/confluence/display/METRON/About+Metron`. Acessado em 04/10/2017.

Carbone, P., Fóra, G., Ewen, S., Haridi, S. e Tzoumas, K. (2015). Lightweight Asynchronous Snapshots for Distributed Dataflows. *Computing Research Repository (CoRR)*, abs/1506.0.

Du, Y., Liu, J., Liu, F. e Chen, L. (2014). A real-time anomalies detection system based on streaming technology. Em *Sixth International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, volume 2, páginas 275–279. IEEE.

Franklin, M. (2013). The Berkeley Data Analytics Stack: Present and future. Em *IEEE International Conference on Big Data*, páginas 2–3. IEEE.

He, G., Tan, C., Yu, D. e Wu, X. (2015). A real-time network traffic anomaly detection system based on storm. Em *Proceedings - 2015 7th International Conference on Intelligent Human-Machine Systems and Cybernetics, IHMSC 2015*, volume 1, páginas 153–156.

Hu, P., Li, H., Fu, H., Cansever, D. e Mohapatra, P. (2015). Dynamic defense strategy against advanced persistent threat with insiders. Em *IEEE Conference on Computer Communications (INFOCOM)*, páginas 747–755.

Jirsik, T., Cermak, M., Tovarnak, D. e Celeda, P. (2017). Toward Stream-Based IP Flow Analysis. *IEEE Communications Magazine*, 55(7):70–76.

Mayhew, M., Atighetchi, M., Adler, A. e Greenstadt, R. (2015). Use of machine learning in big data analytics for insider threat detection. Em *IEEE MILCOM*, páginas 915–922.

Mylavarapu, G., Thomas, J. e TK, A. K. (2015). Real-Time Hybrid Intrusion Detection System Using Apache Storm. Em *17th International Conference on High Performance Computing and Communications*, páginas 1436–1441. IEEE.

Santos, O. (2015). Big data analytics and netflow. Em *Network Security with NetFlow and IPFIX: Big Data Analytics for Information Security*. Acessado em 29/08/2017.

Sanz, I. J., Alvarenga, I. D., Andreoni Lopez, M., Mauricio, L. A. F., Mattos, D. M. F., Rubinstein, M. G. e Duarte, O. C. M. B. (2017). Uma avaliação de desempenho de segurança definida por software através de cadeias de funções de rede. Em *XVII SBSeg 2017*.

Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., Bhagat, N., Mittal, S. e Ryaboy, D. (2014). Storm@Twitter. Em *ACM SIGMOD International Conference on Management of Data*, páginas 147–156. ACM.

Wood, T., Shenoy, P., Venkataramani, A. e Yousif, M. (2009). Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53(17):2923–2938.

Wu, K., Zhang, K., Fan, W., Edwards, A. e Yu, P. S. (2014). RS-Forest: A Rapid Density Estimator for Streaming Anomaly Detection. Em *IEEE International Conference on Data Mining (ICDM)*, páginas 600–609.

Zhao, S., Chandrashekar, M., Lee, Y. e Medhi, D. (2015). Real-time network anomaly detection system using machine learning. Em *11th International Conference on the Design of Reliable Communication Networks (DRCN)*, páginas 267–270. IEEE.