An Evaluation of a Virtual Network Function for Real-Time Threat Detection using Stream Processing

Martin Andreoni Lopez*†, Antonio Gonzalez Pastana Lobato*, Otto Carlos M. B. Duarte*, and Guy Pujolle†

*Grupo de Teleinformática e Automação - Universidade Federal do Rio de Janeiro (COPPE/UFRJ)
Rio de Janeiro, Brazil - Email:{martin, antonio,otto}@gta.ufrj.br
†Laboratoire d'Informatique de Paris 6 - Sorbonne Universities, UPMC Univ Paris 06
Paris, France - Email: {Guy.Pujolle}@lip6.fr

***Abstract*—Network Function Virtualization (NFV) provides new opportunities for efficient and low-cost security solutions. Real-time traffic monitoring and fast security threat detection is a challenge to reduce the risk of great damages. In this paper, we propose a virtualized network function in an Open Source Platform for providing a real-time threat detection service. Our function combines cloud computing and distributed stream processing techniques to accurately and quickly detect threats. The proposed virtualized network function shows a good elasticity shrinking and scaling accordingly to the required load. The results show that the proposed function is able to scale dynamically, analyzing more than five million messages per second. In addition, the function easily migrates sensor elements to reduce latency, allowing the sensor to be located as near as possible to the client.**

I. Introduction

The increase in the volume, velocity and variety of data in current networks demands a robust security infrastructure. Monitoring and processing data at high rates without wasting resources is a huge challenge nowadays. Network usage varies in time and presents several usage peeks, generating many data to be promptly analyzed. The advent of the Internet of Things (IoT) increases the amount of data to be processed. The estimated number of networked sensors by 2025 is around 80 billion [1] and is necessary to guarantee security in this scenario. To face this scenario, automatic and elastic security mechanisms are required. The allocated resources must scale up or shrink to satisfy the real-time requirements to process the traffic and human intervention must be avoided. Unfortunately, conventional centralized solutions are insufficient to ensure security and privacy [2], [3]. Modern advanced models are being proposed to allow real-time distributed stream processing techniques to monitor and analyze traffics of high volume magnitude.

One way to automatize the attack detection accurately is to use machine learning methods. These methods are well suited for big data, since with more samples to train, methods tend to have higher effectiveness [4]. However, there are other aspects to ensure security other than the obtained accuracy in detection. The detection time must be short enough to enable defense mechanisms and the detection infrastructure must be able to process all streaming data in real time, even in scenarios with usage peeks.

We aim to use Network Function Virtualization technology and its cluster infrastructure to combine virtualization, cloud computing and distributed stream processing to detect threats. The objective is to provide an accurate, scalable, and real-time threat detection facility capable to attend usage peeks. The traffic monitoring and threat detection as a virtualized network function presents two main advantages: self-adaptation to different traffic network load and high localization flexibility to place or move network sensors reducing latency.

This paper proposes a virtualized network function in the Open Source Platform for Network Functions Virtualization (OPNFV) that provides an accurate real-time a threat detection service. For the best of our knowledge, this is the first threat detection function implemented in the Open-source Platform for Network Function Virtualization (OPNFV) using machine learning algorithms combined with stream processing. The provided service 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. Besides, the Network Virtualization Platform enables the easy deployment of 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 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 attend the demand.

The remainder of this paper is organized as follow. Section II presents the related work. Section III analyses the Open Source Platform for Network Functions Virtualization.

In Section IV, we present our threat detection virtualized network function. Section V shows the results and Section VI concludes the work.

II. Related Work

Previous works perform intrusion detection in virtualized environments [5], [6], [7], [8]. BroFlow covers the detection and mitigation of Denial of Service (DoS) attacks. Sensors run in virtual machine under Xen hypervisor, and thus includes a mechanism for optimal sensor distribution in the network [5]. An attack mitigation solution, based on Software Defined Networking, complements the proposal, focusing on DoS attacks detection based on an anomaly algorithm implemented in the Bro IDS. Fung and McCormick [6] describe VGuard as one of the first proposals of Distributed Denial of Service (DDoS) mitigation as a Virtual Network Function (VNF). VGuard is a dynamic traffic engineering solution based on prioritization. The solution creates two virtual tunnels, one with higher priority and the other with lower priority for all traffic to the DDoS targeted service. Each flow is allocated to one of these two tunnels to reach the destination and a flow-dispatching algorithm is used to manage the flows based on their priority level. The system is used to detect and mitigate DDoS attack.

Other works use machine learning for attack detection in virtualized environments [7], [8]. Azmandian *et al.* present an application based on machine learning to automatically detect malicious attacks on typical server workloads running on virtual machines. The key idea is to obtain the feature selection by Sequential Floating Forward Selection (SFFS) algorithm, also known as Floating Forward Search, and, then, classify the attacks with the K-Nearest Neighbor (KNN) and the Local Outlier Factor (LOF) machine learning algorithms. The system runs in one physical machine under VirtualBox environment. Li *et al.* present cloudmon [8], a Network Intrusion Detection System Virtual Appliance (NIDS-VA), or virtualized NIDS. Cloudmon enables dynamic resource provisioning and live placement for NIDS-VAs in Infrastructure as a Service (IaaS) cloud environments. The work uses Snort IDS and Xen hypervisor for virtual machine deployment. Moreover, Cloudmon uses fuzzy model and global resource scheduling to avoid idle resources in a cloud environment. The proposal employs the conventional Snort IDS, based on signature method, to detect misuse and focuses on the resource allocation.

We propose a virtualized network function on Open Source Platform for Network Function Virtualization (OPNFV) that provides a threat detection facility. The function employs open source tools to detect threats in real time using flow processing and machine learning techniques.

III. The Open source Platform for Network Function Virtualization

Network Function Virtualization (NFV) technology intends to offer software virtualized network services using costumer on the shelf (COTS) hardware in order to lower Operating Expenditures (OPEX) and Capital Expenditures (CAPEX) costs, and greatly reducing the time to the market (TTM) of innovations [9]. The key idea is to offer communication, processing, and storing service for big data [10]. Thus, Virtual Network Functions (VNF) are implemented in software running on different physical servers, usually on a cluster environment. Therefore, network services such as firewall and threat detection can be executed as a set of VNF allowing a bigger flexibility, scalability, and easier deployment when compared to traditional services. The main goal of the NFV technology is to optimize network services. This concept is complemented with the idea of Software Defined Networking (SDN) that provides a greater programmability for network management due to its global network view in the network controller. Specially, SDN acts in the control and in the implementation of packet forwarding and processing, while NFV acts in the provision of network services, such as firewall, Intrusion Detection System (IDS), Network Address Translation (NAT), or even higher layer services, such as Web servers, email servers, among others.

Instead of using expensive proprietary network equipment, a strong tendency is rising to provide services with open-source software using trusted platforms that integrate the processing, storing, and communication of data. Trying to accelerate the implantation of Virtualized Network Functions, the Linux Foundation develops a collaborative project called Open source Platform for Network Functions Virtualization (OPNFV) [1]. The main idea behind OPNFV is to use open-source software to provide a platform compatible with the European Telecommunications Standards Institute (ETSI) standards.

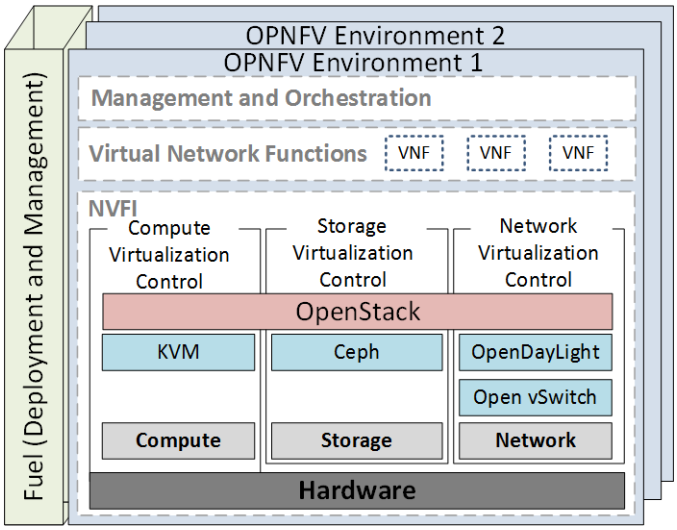

Figure 1: OPNFV architecture, composed by three main components, Network Virtual Function Infrastructure (NFVI), Virtual Network Functions (VNFs), and Management and Orchestration. The deployment and management of the OPNFV environments is coordinated by Fuel.

Figure 1 shows the architecture of the Open source Platform for Network Function Virtualization (OPNFV). Fuel deploys and manages the OPNFV environment. Three main modules

[1]The Open source Platform for Network Functions Virtualization https://www.opnfv.org/

compose this environment, the Network Virtual Function Infrastructure (NFVI), the Virtual Network Functions (VNFs) and the Management and Orchestration (MANO). The NFVI contains the compute, storage and network module. The compute module administrates the virtual machines through the KVM hypervisor. The Storage module uses the Ceph tool that is a distributed object store and file system. The network module uses the Software Defined Networking (SDN) paradigm through the OpenDayLight controller that manages the Open vSwitches virtual switches. Network services are deployed in middleboxes or network appliances called virtualized network functions (VNF). VNFs consist of one or more virtual machines that run specific network functions such as firewall, IDS, NAT, among others. VNFs can be combined together, chaining in deliver full-scale networking communication services [11]. We implemented our Threat Detection System as VNF. Finally, the Management and Orchestration layer provides the logic and functionality required for the provision of resources, configuring the VNFs and the infrastructure.

As already mentioned before, we propose a virtualized network function on Open Source Platform for Network Function Virtualization (OPNFV) that provides a threat detection facility using open source tools. Furthermore, the key idea is use flow processing and machine learning techniques to detect anomalies in real time. The following section presents the open-source tools that are used to obtain real-time threat detection at high processing rate.

IV. The Virtualized Threat Detection Network Function

Real time threat detection is crucial to guarantee network security, because if the detection time is too long, no reaction to neutralize the threat can be effective. To accomplish this goal, our prototype function uses the lambda architecture [12]. Through stream processing and the lambda architecture, it is possible to use parameters from an off-line training for a real-time threat detection. The lambda architecture combines traditional batch processing over a historical database with real-time stream processing analysis. Therefore, the proposed prototype function is able to use trained machine learning techniques to detect both known and zero-day attacks through automatized classification and anomaly detection methods.

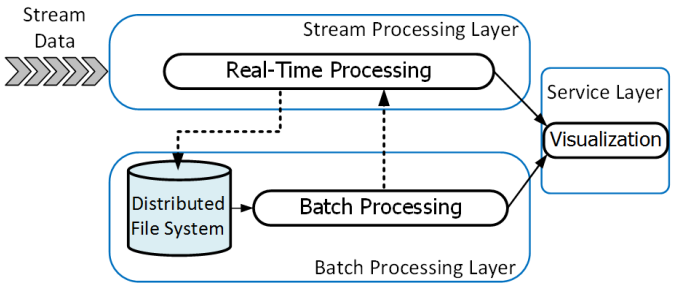

Figure 2: The three-layered lambda architecture, which combines stream with batch processing: stream processing, batch processing, and service layers.

As shown in Figure 2, the lambda architecture has three layers: the stream processing layer, the batch-processing layer, and the service layer. The stream processing layer deals with the incoming data in real-time. The batch-processing layer analyzes a huge amount of stored data in a distributed way through techniques such as map-reduce. Finally, the service layer combines the obtained information of the two previous layers to provide an output composed by analytic data to the user. Therefore, the lambda architecture goal is to analyze, accurately and in real-time, streaming data, even with its ever-changing incoming rate to obtain results in real-time based on historical data.

The proposed threat detection prototype function follows similar design choices of open-source tools that were integrated by the authors in [13]. The prototype is divided in three main modules to perform real-time threat detection. The Capture Module, the Stream Processing Module and the Alarms Module compose our system to ensure security in several network components. Thus, several probes distributed in different network locations compose the Capture Module and, then, the data are grouped to be processed in a centralized point. The network probes are `Bro` sensors that receive networks packets and extract the main features to be further processed. To avoid data losses due to overload, the data is temporary stored in the Apache `Kafka` tool at the analysis location. Basically, `Kafka` is a message broker that works as a publish/subscribe service and acts like a buffer to the processing tool, adapting different generation and processing rates. `Kafka` abstracts the message flow into topics. Producers then write their data into topics from which the consumers can read these data. In the stream Processing Module we use Apache `Spark Streaming` as the stream processing core of the system to detect threats. Finally, once the system detects a threat, it sends an alert to the Alarm Module. This module handles these alerts and it can communicate with another system such as a prevention system in order to manage the threat.

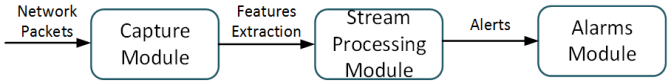

Figure 3: The three-layered lambda architecture, which combines stream with batch processing: stream processing, batch processing, and service layers.

The distributed stream processing tool is the most important module of our network function.Spark is a project initiated by UC Berkeley and is a platform for distributed data processing, written in Java and Scala. Spark has different libraries running on top of the Spark Engine, including Spark Streaming [14] for stream processing. The stream abstraction is called Discrete Stream (D-Stream) defined as a set of short, stateless, deterministic tasks. In Spark, streaming computation is treated as a series of deterministic batch computations on minor intervals. Similar to MapReduce, a job in Spark is defined as a parallel computation that consists of multiple tasks, and a task is a unit of work that is sent to the Task Manager. When a stream enters

Spark, it divides data into micro-batches, which are the input data of the Resilient Distributed Dataset (RDD), the main class in Spark Engine, stored in memory. Then the Spark Engine executes by generating jobs to process the micro-batches.

Spark has streaming delivery semantics of *exactly-once*. The idea is to be able to process a task on various worker nodes. During a fault, the micro-batch processing may simply be recalculated and redistributed. The state of RDDs are periodically replicated to other worker nodes, in case of node failure. Tasks are then discretized into smaller tasks that run on any node without affecting execution. Thus, the failed tasks can be launched in parallel evenly distributing the task without affecting performance. This procedure is called Parallel Recovery. The semantics of *exactly-once* reduces the overhead shown in upstream backup in which all tuples are acknowledge.

In a previous work [15], we showed that Apache Spark Streaming presents better fault tolerant management than other streaming platforms such as Apache Storm and Apache Flink. The fault tolerance feature of Apache Spark Streaming allows our system to process messages even under adverse condition of node failures. This is important for threat detection applications, because it ensures the security of the network even under stress situations. Apache Spark allows defining the parallelism of the application in a way that multiple stream samples can be processed simultaneously.

V. Results

To evaluate the performance of the prototype, we analyze latency requirements and speedup factor for real-time stream processing. We perform the experiments in the OPNFV Danube 2.0 environment and Sahara project for Apache Spark cluster provision. Our OPNFV environment has 96 GB of RAM, 700 TB of storage and 128 cores of Intel Xeon processors with clock frequency of 2.6 GHz.

We calculate the results with 95% of confidence interval. We use the open source software Apache Kafka, version 0.8.2.1, message broker that operates in a publish/subscribe mode, to submit high data rates in the stream processing systems. In Kafka, samples or events are called messages, name that we will henceforth use. Kafka abstracts message stream into topics that act as buffers or queues, adjusting different production to consumption rates. The cluster contains a master node and several workers nodes that are incremented in order to evaluate the system. Each node runs the open source software Apache Spark Streaming version 1.3.1 with 1024 MB of memory RAM and 10 GB of storing.

A dataset is elaborated through the packet capture in computers from our lab, GTA at Federal University of Rio de Janeiro, containing both normal traffic and real network threats [13]. After the packet capture, data from the header, grouped in a time window, generated flow data. We define a flow as a sequence of packets from the same IP source to the same IP destination. Each flow has 24 features, generated by TCP/IP header data, as TCP, UDP and ICMP packet rate, number of source and destination ports, number of each TCP flag, among others. The analysis of packet header information detects two threat classes: Denial of Service (DoS) attacks and probe. Therefore, we elaborate the dataset with several attacks from both these classes. Altogether, the dataset contains seven types of DoS and nine types of probe. The DoS attacks are *ICMP flood*, *land*, *nestea*, *smurf*, *SYN flood*, *teardrop*, and *UDP flood*. The different types of probe in the dataset are *TCP SYN scan*, *TCP connect scan*, *SCTP INIT scan*, *Null scan*, *FIN scan*, *Xmas scan*, *TCP ACK scan*, *TCP Window scan*, and *TCP Maimon scan*. We perform the threats using tools from the *Kali Linux* distribution, which aims to test computer system security. These attacks were labeled in the dataset by origin and destination IP filters, separating the traffic belonging the attack machines from the rest. Altogether, around 95 GB of packet capture data were collected, resulting in 214,200 flows between normal and malicious traffic. The application tested was our threat detection system with a neural network classifier programmed in Java.

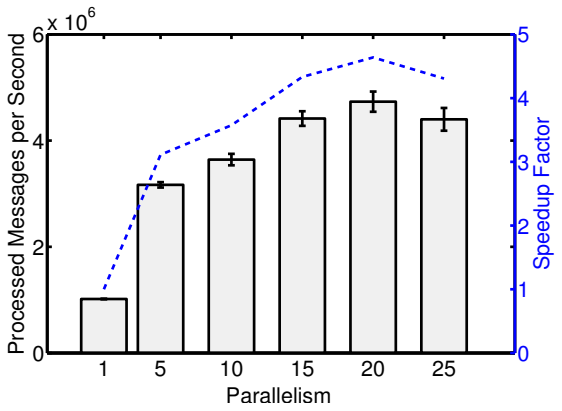

Figure 4: Throughput results of the Apache Spark Streaming Engine in terms of number of messages processed per second in the right axis and latency speedup factor in the left axis. The throughput and latency are in function of the task parallelism.

The first experiment measures the performance of Apache Spark Streaming engine in terms of processing throughput and latency. The data set is injected into the system in its totality and replicated as many times as necessary. The experiment calculates the consumption and processing rate of the stream processing engine. It also varies the parallelism parameter, which represents the total number of cores available for the cluster to process samples in parallel. Figure 4 shows the results of the experiment. In the left y axis it is shown the throughput as the amount of messages processed per minute by the system, and the right y axis indicates the latency comparison in the speedup factor. The speedup factor is calculated as follow: $S_{latency} = \frac{La1}{La2}$; Where $La1$ is the latency of the system when parallelism is equal to one, and $La2$ is the latency of the system with the variation of the parallelism parameter.

The proposed function is able to improve the processing capacity up to twenty cores in parallel. The same behavior is shown for latency as well as for throughput. Considering throughput, the system is able to handle more than five million

of messages per second. Moreover, the latency speed factor with a parallelism of twenty is reached around 4.65, this indicates that the system can parallelize almost five times with twenty virtual machines running one core each.

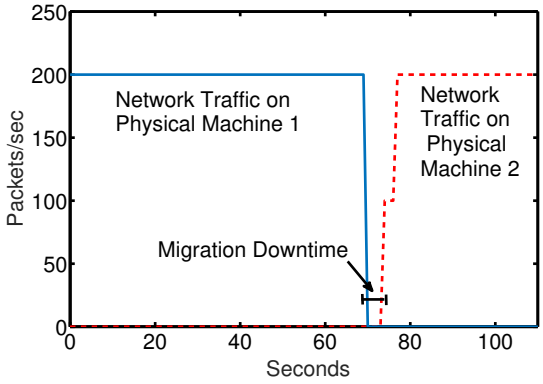

Figure 5: Virtual Machine Migration from the Physical Machine 1 to Physical Machine 2. The constant flow rate applied to the virtual machine at Physical Machine 1, after 60 seconds approximately of migration, goes to Physical Machine 2.

The second experiment aims to show the operation efficiency of the implemented function under live migration. The live migration offers a great flexibility for the user and it is possible thanks to the virtualization, achieved through the OPNFV platform. In our threat detection virtualized network function, live migration provides several advantages. A security advantage is the possibility to place and rearrange dynamically sensor machines to better protect the network and reduce the threat detection time. A general advantage concerning the processing cluster is the ability to migrate machines, allowing a smart distribution among the physical servers and enabling the optimization of the number of running servers, avoiding the waste of resources. Figure 5 shows the behavior of a network flow under live migration. In this experiment, we send a constant rate flow of 200 packets per seconds from one virtual machine to another. Both virtual machines are hosted in the same physical machine. Then, approximately at 60 seconds, the migration process is started, in order to migrate the virtual machine that receive the flow to another physical server. The Figure 5 show the migration downtime is small, making the flow unaffected under the migration. The migration feature, allows our threat detection application to set monitoring sensors as close to the client as possible, avoiding latency problems.

VI. Conclusion

This paper presents the design and the performance evaluation of an efficient threat detection function, deployed as a virtualized network function. For the best of our knowledge, this is the first threat detection function implemented in the Open-source Platform for Network Function Virtualization (OPNFV). The threat detection is performed using machine learning algorithms combined with stream processing. As a use case, we implemented a neural network for attack classification on the top of the Apache Spark Streaming. Our proposed threat detection function presents a high performance and the live migration feature. The results shown that the implemented function prototype presents a high throughput and a low latency. Our function is able to easily scale, and to process more than five million messages per second. In addition, the user is able to migrate the processing VMs and the sensor elements, in order to place the sensors as close to the client as possible, avoiding latency problems.

Acknowledgment

This research is supported by CNPq, CAPES, FAPERJ, and FAPESP (2015/24514-9, 2015/24485-9, and 2014/50937-1).

References

[1] P. Clay, "A modern threat response framework," *Network Security*, vol. 2015, no. 4, pp. 5–10, 2015.

[2] D. S. Terzi, R. Terzi, and S. Sagiroglu, "A survey on security and privacy issues in big data," in *10th International Conference for Internet Technology and Secured Transactions (ICITST)*, Dec. 2015, pp. 202–207.

[3] B. Thuraisingham, "Big data security and privacy," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '15. ACM, 2015, pp. 279–280.

[4] M. Mayhew, M. Atighetchi, A. Adler, and R. Greenstadt, "Use of machine learning in big data analytics for insider threat detection," in *IEEE Military Communications Conference, MILCOM*, Oct. 2015, pp. 915–922.

[5] M. Andreoni Lopez, D. M. Ferrazani Mattos, and O. C. M. B. Duarte, "An elastic intrusion detection system for software networks," *Annals of Telecommunications*, pp. 1–11, 2016.

[6] C. J. Fung and B. McCormick, "VGuard: A distributed denial of service attack mitigation method using network function virtualization," in *11th International Conference on Network and Service Management (CNSM)*. IEEE, 2015, pp. 64–70.

[7] F. Azmandian, D. R. Kaeli, J. G. Dy, and J. A. Aslam, "Securing virtual execution environments through machine learning-based intrusion detection," in *25th International Workshop on Machine Learning for Signal Processing (MLSP)*, 2015, pp. 1–6.

[8] B. Li, J. Li, and L. Liu, "CloudMon: a resource-efficient iaas cloud monitoring system based on networked intrusion detection system virtual appliances," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 8, pp. 1861–1885, 2015.

[9] W. Yang and C. Fung, "A survey on security in network functions virtualization," in *IEEE NetSoft Conference and Workshops*. IEEE, 2016, pp. 15–19.

[10] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262.

[11] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *Transactions on Network and Service Management*, vol. PP, no. 99, May 2016.

[12] N. Marz and J. Warren, *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2013.

[13] A. P. Lobato, M. A. Lopez, and O. C. M. B. Duarte, "An accurate threat detection system through real-time stream processing," Grupo de Teleinformática e Automação (GTA), Universidade Federal do Rio de Janeiro (UFRJ), Tech. Rep. GTA-16-08, 2016.

[14] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," in *XXIV ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 423–438.

[15] M. Andreoni Lopez, A. P. Lobato, and O. C. M. B. Duarte, "A performance comparison of Open-Source stream processing platforms," in *to appear in 2016 IEEE Global Communications Conference (Globecom)*, Washington, USA, Dec. 2016.