A Resilient Stream Learning Intrusion Detection Mechanism for Real-time Analysis of Network Traffic

Eduardo Viegas[1,2], Altair Santin[1], Nuno Neves[2], Alysson Bessani[2], Vilmar Abreu[1]
[1]Graduate Program in Computer Science / Pontifical Catholic University of Parana, Curitiba, Parana, Brazil
[2]LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Lisboa, Portugal
{eduardo.viegas, santin, vilmar.abreu}@ppgia.pucpr.br, {nfneves, anbessani}@ciencias.ulisboa.pt

***Abstract*—The number of novel attacks observed in networked systems increases every day. Due to the large amount of generated data over the network, its storage for further analysis may not be feasible. Moreover, current attacks are becoming more sophisticated, as the attackers are attempting to evade traditional intrusion detection mechanisms by perverting their properties. This paper presents a novel real-time (ongoing) network traffic measurement approach that supports resilient analysis for stream learning intrusion detection. The network data is grouped at runtime according to its characteristics, while each network traffic flow is discretized at regular time intervals. Each network flow is classified by a multi-view stream learning classifiers pool, defining the network flow class through a majority voting approach. The proposal is able to provide resiliency to the classifiers even for the detection of unknown attacks. The evaluation tests for the average operation point (25 views) provides an increase in the system resilience to adversarial attacks of 22 % when compared to traditional approaches. Moreover, in the scalability experiments with a 10-node (single core each) cluster testbed, the network flow measurement solution (1 view) reached 1.38 Gbps throughput, while the proposed resilient stream learning intrusion detection with 25 views reached a throughput of 1.19 Gbps.**

Keywords—Stream processing, adversarial machine learning, stream learning, intrusion detection.

I. INTRODUCTION

According to the Cisco network forecasting report [1], the average broadband speed will nearly double until 2020. Nonetheless, in 2016 the annual global traffic already exceeded the zettabyte threshold and it is expected that until 2020 it will double, when the network traffic originated from smartphones should surpass the one coming from desktop PCs. As the amount of network traffic data increases, it becomes more difficult to collect and analyze the network activity [2].

Current approaches for network traffic measurement in the Big Data context rely on Hadoop-based clusters [3, 4]. These approaches, in general, simply write the raw network traffic activity log (PCAP) to the filesystem (e.g. HDFS [3]) for further analysis. Although such approaches offer significant improvements in network traffic measurement throughput [3], they lack applicability in real-world environments. For production usage, the network traffic must be analyzed as soon as packets are captured to allow proper decision making, e.g., to perform network-based intrusion detection. Moreover, due to the large amount of network data, it is normally impossible to log and store the traffic activity [2].

On the other side, the number of vulnerabilities increases in a daily basis according to a Symantec report [5]. In 2015, 78 percent of the scanned websites were vulnerable, and 15 percent of those were critical. Moreover, in 2015, 12 new vulnerabilities were discovered on average each day [5]. These numbers highlight the need for security mechanisms that can detect previously unknown threats.

Machine learning approaches can be employed for the detection of intrusion attempts, yielding promising results [6]. However, in recent years, some attention has been brought to the usage of machine learning techniques in adversarial settings [7]. In such settings, the attacker will attempt to evade the detection mechanism, either by perverting the properties of the mechanism or by injecting misclassified instances during the training stage [8]. Thereby, it is fundamental for machine learning system employed in intrusion detection to be resilient (often referred as model resilience) to adversarial attacks [7].

Due to the evolving nature of network environments, a great amount of research was carried out on the development of stream learning techniques [9]. Such techniques are often employed in scenarios in which the set of target concepts (classes) changes over time [10]. For instance, in the network-based intrusion detection context, the attacker behavior (intrusion) may change due to the insertion of new attacks [6]. Therefore, in such evolving scenario, the detection mechanism update could be performed at each new event arrival [9]. However, this approach relies in supervised learning, in which the event need to be previously classified [11]. Thereby, rendering traditional stream learning techniques not applicable to real networked environments [6].

This paper presents a network traffic measurement solution for resilient stream learning attack detection for real world network environments. Our proposal continuously processes streams of raw network events in runtime and in a distributed manner. Each network flow is grouped according to its characteristics, and is discretized in time windows for further network traffic classification. In addition, we propose a resilient stream learning intrusion detection technique that addresses adversarial settings. Our proposal relies on a semi-supervised approach to generate a multi-view pool of stream learning decision trees (random forest). The classifiers pool is obtained through a single supervised dataset, and new attacks are reliably learned in an unsupervised manner. The resiliency to adversarial attacks is provided by leveraging from a voting mechanism before the incremental model update, thereby ensuring that each classifier is updated according to the classifiers pool outcome.

The contribution of our work is threefold. First, to the best of our knowledge, this is the first work to address real-time network traffic measurement in the Big Data context, without requiring changes in the stream processing frameworks. This is achieved without storing any data to perform network flow measurement. Second, we propose and evaluate a new approach to provide resiliency to stream learning algorithms in adversarial contexts, while still being able to reliably update the detection system in an unsupervised manner in face of new attacks. Third,

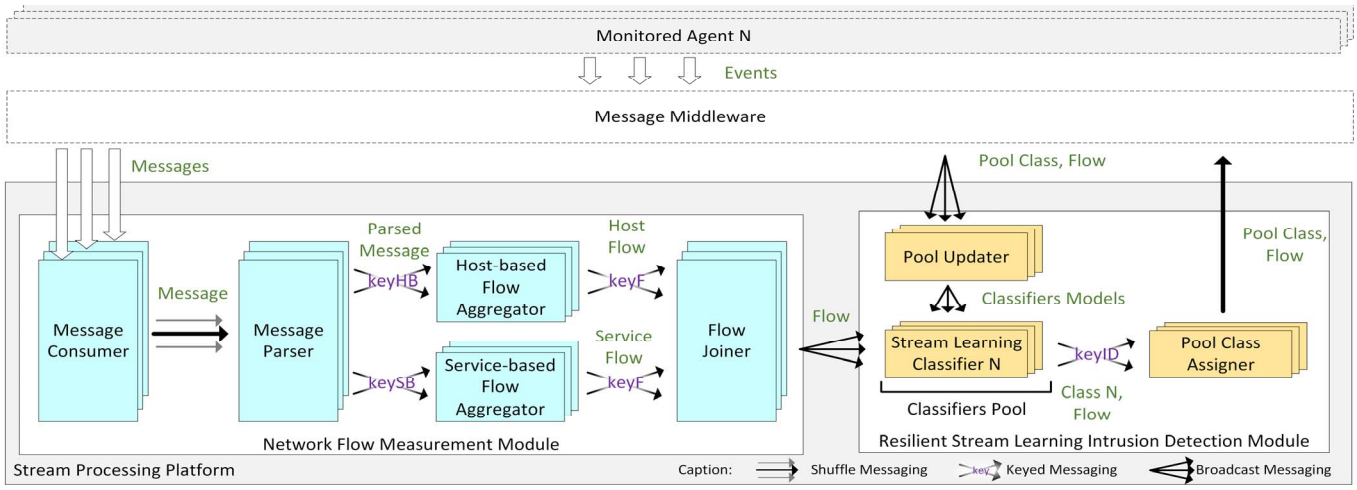

Figure 1 – Overview of the proposed real-time network traffic measurement architecture.

we design and evaluate the port of a stream learning algorithm to stream processing frameworks in terms of accuracy and throughput.

II. Background

In the literature, the approaches for network traffic measurement and classification rely only on the information available in the network packets [6]. The network traffic measurement step is responsible for extracting a set of features from the network data. In general, those features are based in the communication flow, which can be of two types: host-based and service-based.

The host-based features refer to historical attributes regarding the communication (flow) between two hosts, while the service-based refer to historical attributes of the communication between two or more services within those two hosts. The feature set used in this work follows a set of well-known practices suggested by Viegas et al. [12]. A total of 20 features are extracted for each communication flow based on [12]. For instance, an example of a service-based feature is the *num_bytes_src_dst*, which counts the total number of bytes sent from a client to a server. After the feature extraction from the communication flow, the network traffic classification becomes possible.

Several works have been proposed in the literature for network traffic classification [6], even in the Big Data context [4, 13]. In general, the network traffic is classified through machine learning techniques [6], which can be performed through supervised, semi-supervised or unsupervised learning [11]. In supervised learning, all events are correctly classified (labeled) prior to their usage in the training stage, while in unsupervised learning there is no event labeling [11]. The semi-supervised approach relies on a subset of labeled instances, that are used during the initial training stage.

An emerging field of research, known as adversarial machine learning [7], considers that the adversary (attacker) will attempt to evade the intrusion detection mechanism using sophisticated types of attacks, divided in *causative* and *exploratory* [8]. The causative attacks refer to attacks that occur during the classifier training process [8], e.g., the attacker inject malicious packets into the training dataset as normal events. On the other hand, the exploratory type aims at exploring the machine learning algorithm [8], e.g., by designing the attack in a manner that the detection engine classifies it as a normal activity. Adversarial attacks against stream learning algorithms could render the system unreliable, as the learning algorithm will be updated by misclassified instances.

III. Proposal

In this section, we describe the real-time network flow measurement and the resilient stream learning intrusion detection modules.

A. Network Flow Measurement

The architecture of the real-time (ongoing) network traffic measurement system is shown in the left side of Figure 1. A set of monitored agents, e.g. hosts, network switches or routers, transmit the events through a message middleware. An event corresponds to a unit of analysis, such as network packets or netflow records. The message middleware acts as a broker of events, being responsible to provide a single interface for all monitored agents.

The *message consumer* acts as the data producer for the proposed system. Its only purpose is to receive the arriving events in the message middleware, regardless of their content or source agent. Each read event is forwarded to the *Message Parser* in a shuffle manner. The message parser in turn, determines the event fields and type.

The *host-based* and *service-based flow aggregator* modules perform the actual network flow statistics measurements. To do that in runtime and in a distributed manner, both aggregators receive messages through a *keyed* stream (operators receive messages according to a hash function). The key for the host-based flow aggregator is calculated using Eq. 1, whilst the key for the service-based flow aggregator employs Eq. 2, where *src* denotes the event source address and *dst* denotes the event destination address.

$$keyHB = hash(src_{address}) \oplus hash(dst_{address}) \quad (1)$$

$$keySB = hash(src_{address}, src_{port}) \oplus hash(dst_{address}, dst_{port}) \quad (2)$$

The host-based flow aggregator receives events grouped according to the *xor operation* of both source and destination addresses, whilst the service-based flow aggregator also uses the

TCP/UDP port addresses. Through *xor*'ing of the hash values, it is possible to forward messages from two specific hosts (and their services) to the same flow aggregator, regardless of the current flow message source address. Thereby, allowing the computation of messages exchanges between two hosts and their services.

To compute feature values from the grouped events, our solution discretizes them in time intervals, referred as the *tumbling window*. Each tumbling window stores and updates the features values according to each received event. When a tumbling window expires, the flow features values are exported in a host flow or service flow format, and the flow feature values computation starts over again.

For instance, consider two services exchanging messages over the network for 3 seconds, and a tumbling window interval of 2 seconds. To compute those flow values, the message parser module sends all events exchanged between these two services (Eq. 1 and Eq. 2) to the same host-based and service-based flow aggregators. Each aggregator computes the flow features values for the first 2 seconds, and when the tumbling window expires, it exports the host flow and service flow to the next module. When a new event arrives after the initial 2 seconds, the host-based and service-based flow aggregators creates another tumbling window and start the flow features values computation again.

Finally, the *Flow Joiner* module is responsible to receive all host and service flows values, and join them in a single flow. The module receives the exported events through Eq. 3.

$$keyF = hash(src_{address}) \oplus hash(dst_{address}) \quad (3)$$

Thus, the flow joiner module receives all flow values from the same hosts, regardless of the service. For each received service flow, the flow joiner aggregates it to the last exported host flow and exports the result to the next module.

One may note that a single host may have several exported service flows, while having a single host flow, e.g. a single host accessing two services in another host. Thereby, the flow joiner must also store the host flow, to mix it with several exported service flows. To this end, the flow joiner also stores exported host flows in a tumbling window.

B. Resilient Stream Learning Intrusion Detection (SLID)

After the network flow computation and its export, it becomes possible to classify it either as normal or attack. The classification of network flows in evolving and high-speed networks demands resilient, self-updateable and fast classification approaches. To provide such properties, our solution relies in multi-view stream learning decision trees (pool of decision trees, in which each tree is trained with a different subset of features, also known as random forest). The classifier pool is obtained through a single supervised dataset, and new attacks are reliably learned in an unsupervised manner. The overall process is shown in the right side of Figure 1.

In order to setup the initial classifiers pool, our approach relies on a supervised dataset. The pool is obtained through a multi-view approach, in which each classifier is trained with a different subset of features from the supervised dataset.

During production operation, each classifier assigns a class (normal or attack) to the network flows according to the model (learned labeled event). However, the classifier does not incrementally update its model, it simply forwards the network flow and its assigned class to a *Pool Class Assigner* module. This module receives a subset of the exported flows (Eq. 4), in which the $flow_{id}$ is a unique flow identifier.

$$keyID = flow_{id} \quad (4)$$

The pool class assigner module in its turn defines the flow class according to a majority voting scheme from the classes reported by each individual classifier (view). When the flow is properly classified, the module checks whether there was a discrepancy in the classifiers class assignment. A discrepancy is found when the assigned class was not unanimous, in such cases, the module sends a message to the message middleware in order to independently update the classifiers pool, through the *Pool Updater* module.

The pool updater module reads flows to be updated through the message middleware, updates its pool and sends the updated classifiers model for the classification of new flows.

Our proposal is resilient to adversarial attacks since it relies on the multi-view classification, an attacker must change the behavior in a way that the majority of the classifiers, takes the wrong decisions, even though each of them has a unique event view. Nonetheless, due to the distributed nature of stream processing frameworks, we can significantly increase the number of classifiers (views) in our pool without introducing significant overhead, as will be shown in Section VI.C. Moreover, our independent update technique (pool updater) is able to increase the classification system throughput through the separation of the classifiers update and the classification process.

IV. Prototype

Our prototype uses monitored agents deployed in hosts. Each monitored host exports the network packet headers to the message middleware, which, was implemented with the well-known open-source Apache Kafka [16], version 0.10.2.0.

We developed the traffic measurement mechanism using the Apache Flink stream processing framework [15], version 1.2.0. The proposed windowing mechanisms (*Tumbling Window*, Section III.A) was implemented by the Flink API, which provides native windowing support. The customized keyed messaging equations (Eq. 1, Eq. 2, Eq. 3 and Eq. 4) were built using the *KeySelector* Flink interface. The Apache Kafka messages are read through the Apache Flink connector API, version 0.10_2.10.

Each SLID classifier is obtained through the Massive Online Analysis (MOA) API [17], release 16.04. The subset of features (view) used for each SLID classifier is randomly obtained using the same random seed for all evaluation tests at startup time. The parallelism level (number of threads) varies for each experiment, according to the used number of worker nodes. However, for the SLID classifier and pool updater modules, the parallelism level is set according to the number of used SLID classifiers. Each module thread holds a SLID classifier view.

V. Dataset

A testbed environment was deployed to evaluate the prototype, using the approach proposed in [12] to obtain the network traffic. The dataset is composed by two classes of network traffic: *normal* and *attack*. Two attacks categories were considered: *Denial-of-Service* (DoS) and *Probing* (Probe). The

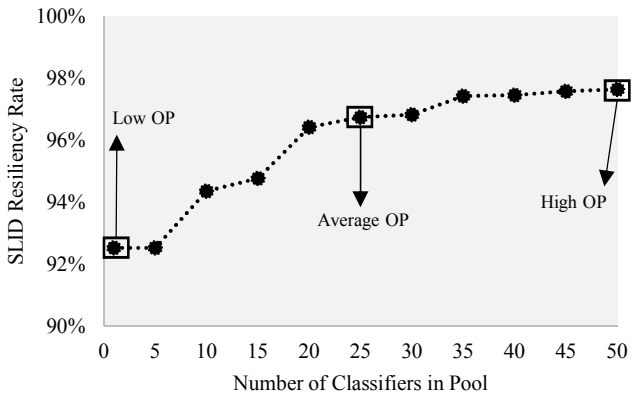

Figure 2 – Tradeoff between SLID resilience rate and classifiers in pool.

following DoS attacks were used: *SYNFlood*, *ICMPFlood*, *UDPFlood*, *HTTPFlood*, *SMTPFlood* and *SlowLoris* (*SYNFlood* attack that keeps the communication alive). As for Probing, the following attacks were trigged: *SYNScan*, *UDPScan*, *NULLScan* and *TCPConnect*.

A honeypot hosts every service (server) as well as receives every attack. The offered services were: HTTP, SSH, SMTP, SNMP and DNS. Each client requests a service through a specific workload tool. The time between each request is pseudo-random, varying from zero to four seconds, to mimic the client unpredictable behavior while requesting a service.

The testbed consists of 100 interconnected clients, each running in a separate Ubuntu 16.10 machine. A single honeypot server was used (honeyd 1.5c), installed on an Ubuntu 16.10. The attacker hosts were running a Kali Linux version 2.0. We used 10 machines to generate the attacks, each attacker host generated a single attack type. The described testbed generated 10 hours' worth of data with more than 615 million network packets in a 175.7 GB PCAP (raw network packet) file.

VI. Evaluation

The evaluation was performed in three steps. First, our proposed *SLID* was tested for accuracy and resiliency. Second, we evaluated our *Network Flow Measurement* mechanism in several configurations of the number of used workers and tumbling window intervals. Finally, we analyzed the whole solution for different *Resilient SLID* operation points and identifying possible performance bottlenecks.

A. Resilience of Stream Learning Intrusion Detection

To the best of our knowledge, there are no standard or well-accepted measures for model resiliency to adversarial attacks yet [7]. Thereby, in this paper, we measure the resiliency through Eq. 5, where N_{update} denotes the total number of samples used to update the detection system in production, while the $Error_{update}$ refers to the total number of misclassified samples from the used set of events to update the detection mechanism.

$$resiliency = \left(1 - \frac{Error_{update}}{N_{update}}\right) x100 \qquad (5)$$

Our proposed resilient SLID relies on a supervised dataset. Therefore, we adopted the following approach: the stream learning classifiers pool is obtained from a supervised dataset containing only *SYNFlood* attacks and normal events, while the resiliency and accuracy rates are defined through the replay of the whole testbed environment (Section V).

TABLE I. Comparison between traditional methods and SLID: Accuracy and Resiliency rates.

Method (%) / Class	SLID (OP, figure 2)			Traditional Machine Learning (TML)	Traditional Stream Learning (TSL)
	Low	Average	High		
Normal	93.93	95.73	96.62	97.96	92.76
SYNFlood	100.00	100.00	100.00	100.00	100.00
ICMPFlood	97.56	100.00	100.00	0.00	97.56
UDPFlood	100.00	100.00	100.00	0.00	100.00
HTTPFlood	100.00	100.00	100.00	100.00	100.00
SMTPFlood	44.87	95.51	100.00	15.38	43.89
SlowLoris	41.94	100.00	100.00	26.31	39.76
SYNScan	100.00	100.00	100.00	67.17	100.00
UDPScan	100.00	100.00	100.00	50.00	100.00
NULLScan	100.00	100.00	100.00	66.67	100.00
TCPConnect	100.00	100.00	100.00	74.18	100.00
Resiliency	***92.53***	***96.74***	***97.64***	***74.99***	***90.44***

The adversarial settings was generated by using the detection mechanism – trained only with *SYNFlood* attacks – against the remaining attacks that are unknown. The used set of services are common in real-world environments. Thus, any intrusion detection mechanism must be able to cope with such attacks, regardless of which of them was used in the training stage to obtain the model.

Notice that our evaluation approach can reproduce both types of adversarial attacks (Section II): (i) *causative*: the SLID is updated using incoming instances, therefore the attacker is able to inject misclassified instances during the ongoing training; and (ii) *exploratory*: the attacker is changing the attack behavior to evade the intrusion detection, e.g., generating *HTTPFlood* attacks in a system trained with *SYNFlood* attacks.

In our tests, the evaluated set of classifiers are trained with a subset of 25% of the SYNFlood attacks recorded in the testbed environment (Section V). The same number of normal events are also randomly selected in the training stage.

1) Classifier's Pool Setup and Evaluation

Each of SLID classifier pool view is represented through a Hoeffding Tree (VFDT – *Very Fast Decision Tree*) [19]. The VFDT enables the event classification through a series of IF-THEN statement, while the SLID model can be updated incrementally. Each view is obtained using 50 percent of the original feature set. A tumbling window of 2 seconds was used.

Figure 2 shows the resiliency rate tradeoff while varying the number of views from 1 to 50. It is possible to note a proportional relationship between the number of views and the resiliency rate. For the tests, three Operation Points (OP) were chosen: *Low, Average* and *High*, with 1, 25 and 50 views, respectively.

To compare our proposal, we have also considered two other approaches: *traditional machine learning* (TML) and *traditional stream learning* (TSL). The TML refers to the traditional machine learning approach in which the classifier model is obtained in an offline manner. For the tests the J48

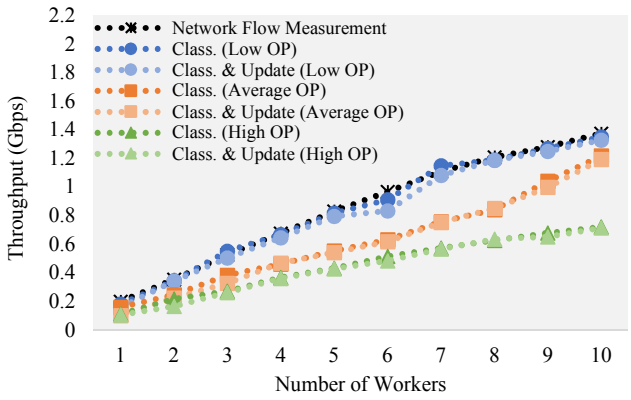

Figure 3 – Operation Points throughput according to the number of Workers.

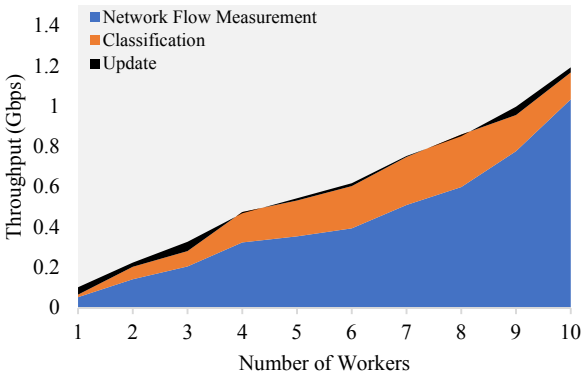

Figure 4 – Performance breakdown for an *Average Operation Point*.

decision tree classifier was used. While the TSL relies in a single VFDT stream learning classifier, using all features, in which all classified instances are fed back to the stream learning algorithm for its incremental update.

Table 1 shows the accuracy and resiliency rates for all considered detection methods. The resiliency rate for the TML refers only to the mechanism accuracy. The proposed *Resilient SLID* approach is better than both of the evaluated techniques, regardless of the considered operation point. TML and TSL had resiliency rates of only 74.99 and 90.44 percent, respectively. On the other hand, our proposal was able to reach resiliency rates of 92.53, 96.74 and 97.64 percent for the *Low*, *Average* and *High* operation points, respectively. Regarding the *Average* operation point (25 views), the proposed approach improved the resiliency rate by 21.75 and 6.30 percent when compared to TML and TSL, respectively. Thereby, our proposed *Resilient SLID* was able to provide a resilient stream learning intrusion detection mechanism.

B. *Network Flow Measurement*

Our prototype was setup on a 12-node cluster in a single rack, connected through a 10 GbE interface. Each node had a single core CPU and 4 GB of memory. A total of 12 workers were used to run the measurement tasks. In all experiments, we have the following scenario: 1 worker executes the Apache Kafka, 1 worker executes the Job Manager and a group of 1 to 10 worker nodes executes the Task Manager, each with a single task slot.

Figure 3 shows the relationship between the system throughput and the number of workers for a tumbling window of 2 seconds. When executing only the *Network Flow Measurement* module, the SLID is able to reach 0.20 Gbps throughput using 1-worker and 1.38 Gbps throughput using 10-worker cluster. Thereby, the increase in average throughput is 0.12 Gbps for each added worker (with a single core).

C. *Network Flow Measurement and Resilient SLID*

The three chosen operation points (Table 1) were evaluated for throughput purposes, whether the classifier pool update is performed or not. Figure 3 shows the system throughput impact when the pool update is performed (*Class. & Update*) and when only the classification (*Class.*) is made.

One can notice that there is a direct relationship between the chosen operation point (*Low, Average* or *High*) and the SLID throughput. In the case of a 10-worker cluster, the *Resilient SLID* presented a throughput of 1.32 Gbps, 1.19 Gbps and 0.71 Gbps for the *Low*, *Average* and *High* operation points, respectively. Therefore, the increase in the resiliency rate incurs in a throughput reduction, due to the increase in the number of classifiers (views). Our proposed independent pool update approach could decrease further the performance impact, by separating the event class assignment process (Figure 1, *Pool Class Assigner*) to the pool update process (Figure 1, *Pool Updater*). Therefore, the pool update process incurred only in a throughput decrease of 5.37 percent.

In average, the *Network Flow Measurement*, *Classification* and *Update* process, represented 67.85, 25.41 and 6.74 percent respectively from the overall demanded computation process (Figure 4). Thereby, the *Resilient SLID* (Classification and Update) module demanded only 32.15 percent from the computation process (on average).

VII. Related Work

Network traffic measurement is a well-established field. However, due to the increase in the amount of generated data by networked systems, a lot of attention has been brought for the measurement of massive network activities [2]. Lee and Lee [3] proposed a Hadoop-based Internet traffic monitoring and analysis system. The authors performed the flow reconstruction by mapping raw network activity (PCAP) files in HDFS. Their proposed approach achieved 14 Gbps in a 200-node (2 cores each) cluster. However, their system required the previous storage of the PCAP files. The authors also performed the network analysis relying in a simple connection threshold through Hive [3] queries, which could be easily evaded by sophisticated attackers.

Fortugne et al. [4] focused in integrating several anomaly-detectors in the Hadoop architecture for network monitoring. The authors also adopted a similar hash function approach to divide network traffic in *splits*. Each *split* had an anomaly-detection algorithm, which identify network activity by their anomalous score according a specific threshold. However, their approach is susceptible to adversarial attacks. Moreover, their reported system throughput is unfeasible for monitoring of fast networks.

Several approaches have been proposed for network monitoring. However, in their majority, they rely in the previous storage of PCAP data. Thus, the usage of Data Stream Management Systems (DSMS) was considered in some works [13, 20]. The DSMS for network monitoring enable to perform continuous monitoring over received data. Baer et al. [13] proposed a Data Stream Warehouse for network monitoring. The authors also relied on time windows for incremental and continuous queries execution. Moreover, they integrated their proposal with a machine learning framework for the classification of exported time windows. However, their approach relied on a supervised dataset, without concerning about the scalability of the machine learning algorithms. The authors also did not address resiliency, neither model updates.

Due to the inherent evolving nature of the environments where machine learning has been applied over the last years, stream learning algorithms have seen an increasing interest from the research community [19]. However, current approaches to perform stream learning in the Big Data context are still incipient. Apache Samoa [21] provides a platform for mining evolving big data streams. In their distributed VFDT implementation, they aim at decreasing the tree model update time by using a vertical parallelism approach, in which every example is split according to its attributes. Thus, model updates can be performed in parallel. Their approach only works well for highly dimensional data. In Storm-Moa [22], a horizontal approach is adopted, in which several stream learning algorithms are distributed and new examples are classified in a round-robin fashion amongst the pool. However, their approach assumes a supervised scenario.

VIII. Conclusion

We presented a resilient stream learning intrusion detection mechanism for measurement and analysis of network traffic in near real-time. The network measurement is performed without storing any data, through purely stream analysis. We employ a tumbling window to compute network communication statistics over time from both hosts and services.

The SLID resiliency to adversarial attacks is effective, while still is able to reliably update the SLID in an unsupervised manner, when facing new attacks. The resiliency is reached in two ways, firstly by generating a pool of classifiers, each one with a unique event view (subset of features), and secondly by relying on a voting mechanism used before the model update, thereby ensuring that each classifier is updated according with the pool of outcome. Finally, to provide a high classification throughput, we update the SLID pool in an independent manner, separating the classification process from the update process.

The evaluation tests showed the proposal feasibility. A 10-worker (with a single core each) cluster was able to achieve 1.38 Gbps throughput during the network measurement evaluation tests, a 0.12 Gbps throughput increase in for each additional worker.

The resiliency rates were 92.53, 96.74 and 97.64 percent for the *Low*, *Average* and *High* operation points, respectively. The *Average* operation point increased the resiliency rate by 21.75 percent when compared to traditional machine learning and 6.30 percent for the traditional stream learning.

Finally, the proposed resilient SLID mechanism was able to achieve a throughput of 1.32 Gbps, 1.19 Gbps and 0.71 Gbps for the *Low*, *Average* and *High* operation points respectively, while the pool model update incurred only in a throughput decrease of 5.37 percent (on average).

Acknowledgment

This work was partially sponsored by Coordination for the Improvement of Higher Education Personnel (CAPES), grant 99999.008512/2014-0, by FCT through projects LaSIGE (UID/CEC/00408/2013) and Resilient Supervision and Control in Smart Grids, and by the European Commission through the H2020 grant agreement 700692 (DiSIEM).

References

[1] Cisco VNI Forecast and Methodology, 2015-2020. Available at: http://www.cisco.com/, [Accessed: September 2017].

[2] R. Zuech, T. M. Khoshgoftaar, and R. Wald, "Intrusion detection and Big Heterogeneous Data: a Survey," *J. Big Data*, vol. 2, no. 1, pp. 1–41, 2015.

[3] Y. Lee and Y. Y. Lee, "Toward scalable internet traffic measurement and analysis with Hadoop," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 1, pp. 5–13, 2012.

[4] R. Fontugne, J. Mazel, and K. Fukuda, "Hashdoop: A MapReduce framework for network anomaly detection.," *INFOCOM Work.*, pp. 494–499, 2014.

[5] Internet Security Threat Report 2016. Available at: https://www.symantec.com/security-center/threat-report, [Accessed: September 2017].

[6] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," *2010 IEEE Symp. Secur. Priv.*, vol. 0, no. May, pp. 305–316, 2010.

[7] P. McDaniel, N. Papernot, and Z. B. Celik, "Machine Learning in Adversarial Settings," *IEEE Secur. Priv.*, vol. 14, no. 3, pp. 68–72, 2016.

[8] J. D. Tygar, "Adversarial machine learning," *IEEE Internet Comput.*, vol. 15, no. 5, pp. 4–6, 2011.

[9] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning," *Proc. - IEEE Symp. Secur. Priv.*, vol. 2015–July, pp. 463–480, 2015.

[10] I. Agrafiotis, A. Erola, J. Happa, M. Goldsmith, and S. Creese, "Validating an Insider Threat Detection System: A Real Scenario Perspective," *SPW - IEEE Secur. Priv. Work.*, pp. 286–295, 2016.

[11] Q. V Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean, and A. Y. Ng, "Building high-level features using large scale unsupervised learning," *Proc. 29th Int. Conf. Mach. Learn.*, pp. 81–88, 2012.

[12] E. Viegas, A. Santin, A. França, R. Jasinski, V. Pedroni, and L. Oliveira, "Towards an Energy-Efficient Anomaly-Based Intrusion Detection Engine for Embedded Systems," *IEEE Transactions on Computers,* vol. 66, no. 1, pp. 1–14, 2016.

[13] A. Baer, P. Casas, A. D'Alconzo, P. Fiadino, L. Golab, M. Mellia, and E. Schikuta, "DBStream: A holistic approach to large-scale network traffic monitoring and analysis," *Comput. Networks*, vol. 107, pp. 5–19, 2016.

[14] Apache Storm. Available at: http://storm.apache.org/, [Accessed: September 2017].

[15] Apache Flink. Available at: https://flink.apache.org/, [Accessed: September 2017].

[16] Apache Kafka. Available at: https://kafka.apache.org/, [Accessed: September 2017].

[17] Massive Online Analysis (MOA) . Available at: http://moa.cms.waikato.ac.nz/, [Accessed: September 2017].

[18] C. Gates and C. Taylor, "Challenging the Anomaly Detection Paradigm: A Provocative Discussion," *Proc. 2006 Work. New Secur. Paradig.*, pp. 21–29, 2007.

[19] P. Domingos and G. Hulten, "Mining high-speed data streams," *ACM SIGKDD*, pp. 71–80, 2000.

[20] A. Bar, A. Finamore, P. Casas, L. Golab, and M. Mellia, "Large-scale network traffic monitoring with DBStream, a system for rolling big data analysis," *2014 IEEE Int. Conf. Big Data (Big Data)*, pp. 165–170, 2014.

[21] Apache SAMOA. Available at: https://samoa.incubator.apache.org/, [Accessed: September 2017].

[22] StormMOA. Available at: https://github.com/vpa1977/stormmoa, [Accessed: September 2017].