Network security and anomaly detection with Big-DAMA, a big data analytics framework

Network Security and Anomaly Detection with Big-DAMA, a Big Data Analytics Framework

Pedro Casas, Francesca Soro, Juan Vanerio, Giuseppe Settanni, Alessandro D'Alconzo
AIT Austrian Institute for Technology
email: firstname.surname@ait.ac.at

***Abstract*—The complexity of the Internet and the volume of network traffic have dramatically increased in the last few years, making it more challenging to design scalable Network Traffic Monitoring and Analysis (NTMA) systems. Critical NTMA applications such as the detection of network attacks and anomalies require fast mechanisms for on-line analysis of thousands of events per second, as well as efficient techniques for off-line analysis of massive historical data. The high-dimensionality of network data provided by current network monitoring systems opens the door to the massive application of machine learning approaches to improve the detection and classification of network attacks and anomalies, but this higher dimensionality comes with an extra data processing overhead. In this paper we present Big-DAMA, a big data analytics framework (BDAF) for NTMA applications. Big-DAMA is a flexible BDAF, capable of analyzing and storing big amounts of both structured and unstructured heterogeneous data sources, with both stream and batch processing capabilities. Big-DAMA uses off-the-shelf big data storage and processing engines to offer both stream data processing and batch processing capabilities, decomposing separate engines for stream, batch and query, following a Data Stream Warehouse (DSW) paradigm. Big-DAMA implements several algorithms for anomaly detection and network security using supervised and unsupervised machine learning (ML) models, using off-the-shelf ML libraries. We apply Big-DAMA to the detection of different types of network attacks and anomalies, benchmarking multiple supervised ML models. Evaluations are conducted on top of real network measurements collected at the WIDE backbone network, using the well-known MAWILab dataset for attacks labeling. Big-DAMA can speed up computations by a factor of 10 when compared to a standard Apache Spark cluster, and can be easily deployed in cloud environments, using hardware virtualization technology.**

Keywords*—*Big-Data; Network Traffic Monitoring and Analysis; Network Attacks; Machine Learning; High-Dimensional Data; MAWILab.

I. Introduction

Network Traffic Monitoring and Analysis (NTMA) is paramount today to shed light on the operation of complex and large networks, especially when things go wrong. A main challenge for big NTMA applications is the analysis and processing of very large amounts of fast and heterogeneous network monitoring data. Network monitoring data usually comes in the form of high-speed streams, which need to be rapidly and continuously processed and analyzed. Different systems have been conceived in the past to collect large amounts of measurements in operational networks. What is needed is a flexible data processing system capable to analyze and extract useful insights from such rich data. The fast development of Big Data processing solutions has boosted the implementation of novel solutions for big data processing. However, conceived platforms are very dissimilar, with different requirements, and are conceived for very specific targets and needs. As a result, each Big Data practitioner needs to muddle through the wide range of available solutions. Similarly for Big Data analytics through machine learning based techniques, there is a big gap to the application of such techniques for NTMA applications, despite the growing existence of ML libraries for Big Data platforms.

In this paper we introduce Big-DAMA, a Big Dara Analytics Framework (BDAF) for NTMA applications, designed with comprehensive network monitoring in mind. Starting from a predecessor system called DBStream [8], we have conceived a first running prototype of the Big-DAMA BDAF. Big-DAMA is a flexible BDAF, capable of analyzing and storing big amounts of both structured and unstructured heterogeneous data sources, with both stream and batch processing capabilities. Big-DAMA implements multiple data analytics algorithms for network security and anomaly detection using both supervised and unsupervised machine learning models. These models are implemented using off-the-shelf machine learning libraries.

The Big-DAMA framework is currently deployed as an experimental cluster running on top of virtual hardware technology. To show the application of Big-DAMA in an operational NTMA application, we apply the Big-DAMA BDAF to the detection of network attacks and traffic anomalies on top of real datasets collected at the WIDE backbone operational network. The principal challenge in automatically detecting network attacks and traffic anomalies is that these are generally evolving targets. It is difficult to precisely and continuously define the set of possible anomalies that may arise, especially in the case of network attacks, because new attacks as well as new variants to already known attacks are continuously emerging. As such, a general anomaly detection system should be able to detect a wide range of anomalies with diverse structures, and ML-based models provide promising results to capture the underlying characteristics of such unexpected events. This paper builds on top of our recent early work (i.e., extended abstracts) on big-data analytics [6] and machine learning for network security [7].

The reminder of the paper is structured as follows. Section II presents an overview on the related work. In Section III we

The research leading to these results has been partially funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-129, "BigDAMA".

describe the main characteristics of the Big-DAMA BDAF. Section IV presents the experimental results of the study, including an in-depth analysis on the application of Big-DAMA to the automatic detection and analysis of network attacks and traffic anomalies, as well as a performance comparison of Big-DAMA against other BDAFs. Finally, Section V concludes the paper.

II. State of the Art

The Big Data era has led to an ever-growing research and development of big data data processing systems [10]. An overview of past and current Big Data Analysis Frameworks includes traditional Database Management Systems (DBMS) and extended Data Stream Management Systems (DSMSs), noSQL systems, and Graph-oriented systems. While most target the off-line analysis of static data, some systems target the on-line analysis of data streams.

DSMSs such as Gigascope [11] and Borealis [12] support continuous on-line processing, but cannot run off-line analytics over static data. The Data Stream Warehousing (DSW) paradigm can to handle both on-line and off-line processing requirements within a single system. DataCell, DataDepot [9] and DBStream [8] are examples of DSWs. NoSQL systems such as MapReduce [13] have also rapidly evolved, supporting the analysis of unstructured data. Apache Hadoop [14] and Spark [15] are very popular implementations of MapReduce systems. These are based on off-line processing rather than stream processing. There has been some promising recent work on enabling real-time analytics in NoSQL systems, such as Spark Streaming [16], Indoop [17], Muppet [18] and SCALLA [19], but these remain unexploited in the NTMA domain. Besides these systems, there is a large range of alternatives, including Storm, Samza, Flink (stream-based analysis); Hawq, Hive, Greenplum (SQL-oriented); Giraph, GraphLab, Pregel (graph-oriented), as well as well known DBMSs commercial solutions such as Teradata, Dataupia, Vertica and Oracle Exadata (just to name a few of them).

The application of BDAFs for NTMA tasks requires particular system capabilities, such as scalability, real-time processing, iii) historical data processing, and iv) the availability of traffic data analysis algorithms. Traditional SQL-like databases are inadequate for the continuous real-time analysis of data. As we mentioned before, DSWs have been introduced to extend traditional database systems with continuous data ingest and processing. These technologies leverage arbitrary SQL frameworks to perform rolling data analysis, i.e., they periodically import and process batches of data arriving at the system. In some cases, such technologies have shown outperforming results when compared to off-the-shelf Big Data technologies [8]. More recent solutions include ENTRADA [23], a Hadoop-based DSW for network traffic analysis, using off-the shelf Impala query engine and Parquet file format based on Google's Dremel [24] to achieve high performance, relying on columnar data storage. BDAFs based on the MapReduce paradigm have been recently started to be adopted for NTMA applications [20]. Considering the specific context of network monitoring, some solutions to adapt Hadoop to process traffic data have been proposed [21]. However, the main drawback of such Big Data technologies in general is their inherent off-line processing, which is not suitable for real-time traffic analysis, highly relevant in NTMA tasks. One of the few systems that leverage Hadoop for rolling traffic analysis is described in [22].

The current trend in BDAFs is towards on-line data processing, using stream-based processing engines such as Spark Streaming, Storm, Samza and Flink, but none of them has been yet applied to the NTMA domain.

III. The Big-DAMA Framework

The main purpose of Big-DAMA is to analyze and store large amounts of network monitoring data. Big-DAMA uses off-the-shelf big data storage and processing engines to offer both stream data processing and batch processing capabilities, following a standard lambda architecture, decomposing separate frameworks for stream, batch and query. Lambda architecture is a data processing architecture designed to handle massive quantities of data by taking advantage of both batch- and stream-processing methods. This approach to architecture attempts to balance latency, throughput, and fault tolerance by using batch processing to provide comprehensive and accurate views of batch data, while simultaneously using real-time stream processing to provide on-line data analysis capabilities.

In a nutshell, Big-DAMA uses Apache Spark streaming for stream-based analysis, Spark for batch analysis, and Apache Cassandra for query and storage. There are two main reasons for using Cassandra instead of simply HDFS or Hadoop-based DBs such as Hive: fault-tolerance and speed. Cassandra is fully distributed and has no single point of failure, whereas HDFS has a single point-of-failure represented by the HDFS name nodes. Regarding speed, Cassandra has been built from scratch for the particular case of on-line transactional data, whereas HDFS follows a more static data warehousing perspective. In addition, Cassandra is highly scalable and provides linear scalability without compromising processing performance. Finally, being a NoSQL system it allows to store and handle multiple sources of heterogeneous data, including unstructured data.

Fig. 1 shows a high level architectural design og Big-DAMA. Inspired on DBStream [8], the Big-DAMA BDAF follows a DSW paradigm, offering the possibility of combining on-the-fly data processing with large-scale storage and analytic capabilities. This paradigm provides the means to handle both types of on-line and off-line processing requirements within a single system.

For the purpose of anomaly detection, Big-DAMA implements the Automatic Network Anomaly Detection and Diagnosis system (ANAD^2) we previously conceived in [5]. Following this framework, network traffic is processed on the fly and two different types of features are extracted: symptomatic features and diagnostic features. All features are analyzed for the sake of anomaly detection. However, the symptomatic features are designed such that their changes directly relate to the presence of abnormal and potentially harmful events. On the other hand, changes in the diagnostic features per-se do not have a negative connotation, but rather ease and guide the interpretation of the anomalous event. Within the Big-DAMA BDAF, we have conceived different algorithms for network security and anomaly detection using supervised and unsupervised machine learning models. These models are currently implemented on top of the Big-DAMA batch-

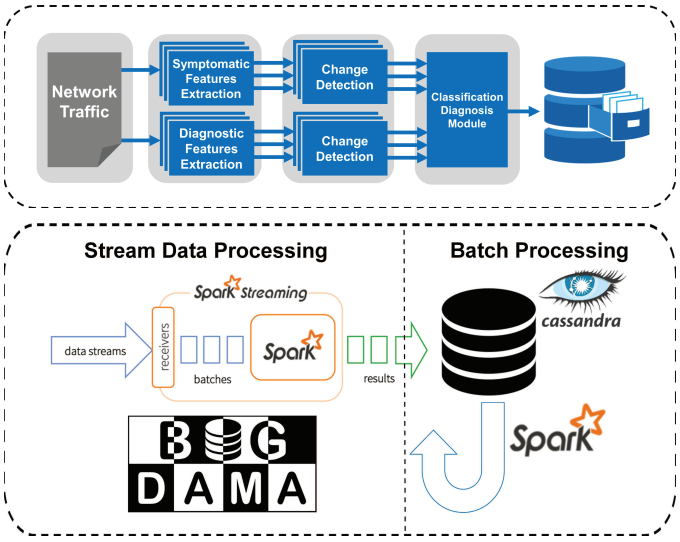


Figure 1: (top) Automatic Network Anomaly Detection and Diagnosis system - ANAD[2]. (bottom) Data Stream Warehouse-based architecture for Big-DAMA platform, using Hadoop ecosystem. ANAD[2] runs on top of Big-DAMA.

processing branch, using off-the-shelf, Spark ML machine learning libraries.

Following the recently introduced data flow model [25], we are currently exploring the adoption of Apache Beam, an advanced unified programming model, which simplifies the implementation of both batch and streaming data processing jobs which can run on the base Spark execution engine used by Big-DAMA, by offering a single unified programming model.

Big-DAMA is currently deployed on top of a virtualized data cluster, consisting of 12 virtual nodes with a total capacity of 150 GB of memory and 30 TB of data storage, connected through Open vSwitch technology. The physical infrastructure consists of 3 physical server nodes, each equipped with 2 Intel Xeon(R) E5-2630 v2-2.60GHz CPUs (24 cores total) and with a total capacity of 64 GB of memory. Virtualization is achieved by Linux Kernel-based VMs, using Proxmox Virtual Environment (https://www.proxmox.com/) for management and orchestration. Big data frameworks are partially managed through a Cloudera, Hadoop ecosystem installation (https://www.cloudera.com/), using distribution CDH 5.10 and Cloudera Manager (with Spark 2).

IV. Detecting Network Attacks with Big-DAMA

To showcase Big-DAMA, we apply the system to the analysis of diverse types of network attacks on real network traffic measurements collected at the WIDE backbone network, using the well-known MAWILab dataset for attacks labeling [1]. MAWILab is a public collection of 15-minute network traffic traces captured every day on a backbone link between Japan and the US since 2001. Building on this repository, the MAWILab project uses a combination of four traditional anomaly detectors (PCA, KL, Hough, and Gamma, see [1]) to partially label the collected traffic.

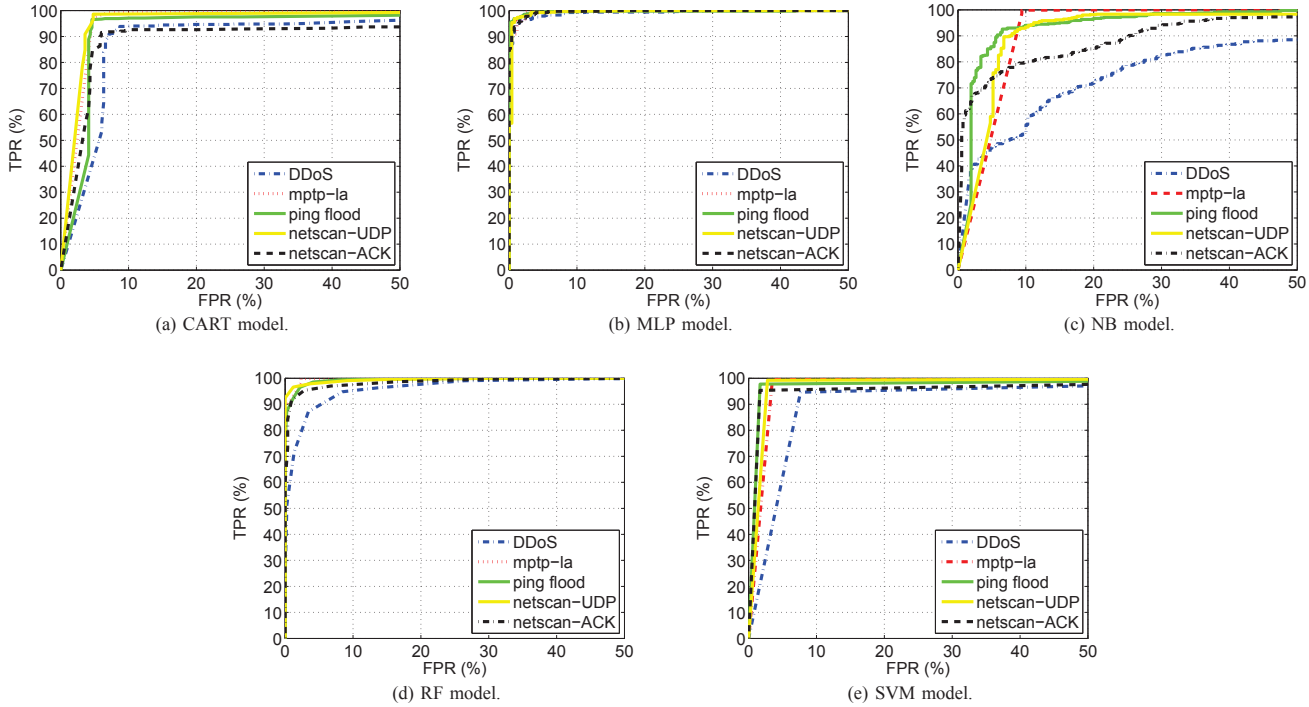
We firstly evaluate the detection performance and execution times of five different supervised ML models, using as input a very large number of traffic features extracted from the packet traces. We then study in detail the relevance of the different extracted features to detect the analyzed attacks, and evaluate different feature selection approaches to keep the most relevant features and improve execution times. Finally, we benchmark the performance of Big-DAMA against other big data platforms, to show that the proposed system is not only highly accurate but can also outperform other similar platforms in terms of execution times. We firstly evaluate the detection performance and execution times of five different supervised ML models, using as input a very large number of traffic features extracted from the packet traces. We then study in detail the relevance of the different extracted features to detect the analyzed attacks, and evaluate different feature selection approaches to keep the most relevant features and improve execution times. Finally, we benchmark the performance of Big-DAMA against other big data platforms, to show that the proposed system is not only highly accurate but can also outperform other similar platforms in terms of execution times.

Table I: Input features for the ML-based detectors.

Field	Feature	Description
Tot. volume	# pkts	num. packets
	# bytes	num. bytes
PKT size	pkt_h	H(PKT)
	pkt_{min,avg,max,std}	min/max/std, $\overline{\text{PKT}}$
	pkt_p{1,2,5,...95,97,99}	percentiles
IP Proto	# ip_protocols	num. diff. IP protocols
	ipp_h	H(IPP)
	ipp_{min,avg,max,std}	min/max/std, $\overline{\text{IPP}}$
	ipp_p{1,2,5,...95,97,99}	percentiles
	% icmp/tcp/udp	share of IP protocols
IP TTL	pkt_h	H(TTL)
	ttl_{min,avg,max,std}	min/max/std, $\overline{\text{TTL}}$
	ttl_p{1,2,5,...95,97,99}	percentiles
IPv4/IPv6	% IPv4/IPv6	share of IPv4/IPv6 pkts.
	# IP_src/dst	num. unique IPs
	top_ip_src/dst	most used IPs
TCP/UDP ports	# port_src/dst	num. unique ports
	top_port_src/dst	most used ports
	port_h	H(PORT)
	port_{min,avg,max,std}	min/max/std, $\overline{\text{PORT}}$
	port_p{1,2,5,...95,97,99}	percentiles
TCP flags (byte)	flags_h	H(TCPF)
	flags_{min,avg,max,std}	min/max/std, $\overline{\text{TCPF}}$
	flags_p{1,2,5,...95,97,99}	percentiles
	% SYN/ACK/PSH/...	share of TCP flags
TCP WIN size	win_h	H(WIN)
	win_{min,avg,max,std}	min/max/std, $\overline{\text{TCPF}}$
	win_p{1,2,5,...95,97,99}	percentiles

A. Data Description & ML Models

The traffic studied in this paper spans two months of packet traces collected in late 2015. From the labeled anomalies and attacks, we focus on a specific group which are detected simultaneously by the four MAWILab detectors, using in particular those events which are labeled as “anomalous” by MAWILab. We consider in particular 5 types of attacks/anomalies: (i) DDoS attacks (DDoS), (2) HTTP flashcrowds (mptp-la), (3) Flooding attacks (Ping flood), and two different flavors of distributed network scans (netscan) using (4) UDP and (5) TCP-ACK probing traffic. We train different ML models to detect each of these attack types separately, thus each detection



(a) CART model. (b) MLP model. (c) NB model.

(d) RF model. (e) SVM model.

Figure 2: Detection performance per type of attack and ML-based approach.

approach consists of five different detectors which run in parallel on top of the data, each of them specialized in detecting one of the five aforementioned attacks types. As a result, each detection approach can not only detect the occurrence of an attack, but also classify its nature.

We select five fully-supervised models to conceive the detectors, including CART Decision Trees (CART), Random Forest (RF), Support Vector Machines (SVM), Naïve Bayes (NB) and Neural Networks (MLP). We selected these detectors based on the a-priori good performance shown by their application in previous work on anomaly detection [2] and traffic classification [3]. We use Spark ML Machine-Learning libraries to calibrate these ML-based algorithms and to perform the evaluations. We address the interested reader to the survey [3] and to the Spark ML documentation for additional information on the algorithms and their configuration parameters. We evaluate the detection performance per anomaly type, considering a slotted, time-based evaluation. For doing so, we split the traffic traces in consecutive time slots of one second each, and compute a set of features describing the traffic in each of these slots. In addition, each slot i is assigned a label l_i, consisting of a binary vector $l_i \in \mathbb{R}^{5\times 1}$ which indicates at each position if anomaly of type $j = 1..5$ is present or not in current time slot.

To better detect the attacks and anomalies, we compute a large number n of features describing a time slot, using traditional packet measurements including traffic throughput, packet sizes, IP addresses and ports, transport protocols, flags, etc. Tab. I describes the set of $n = 245$ features, which are computed for every time slot $i = 1..m$. Note that besides using traditional features such as min/avg/max values of some of the input measurements, we also consider the empirical distribution of some of them, sampling the empirical distribution at many different percentiles. This provides as input much richer information, as the complete distribution is taken into account. We also compute the empirical entropy $H(\cdot)$ of these distributions, reflecting the dispersion of the samples in the corresponding time slot.

B. Detection Performance with Full Features

We test the detection/classification capabilities of the five supervised approaches by computing the True and False Positive Rates (TPR/FPR) for each of the attack types, using as input the full set of 245 features. Fig. 2 depicts the Receiver Operating Characteristic (ROC) curves obtained with each detector, for the proposed attack classes. To reduce over-fitting, all presented results correspond to 10-fold cross validation. Fig. 2 provides the comparative results obtained for the selected supervised detectors. Besides the NB model, the tested approaches provide all highly accurate results for the five types of attacks. In general, detection performance is slightly worse for DDoS attacks. Both the MLP and the RF models achieve the best performance, detecting around 80% of the attacks without false alarms.

Tab. II reports, for each model, both the area under the ROC curve and the total execution time for the complete 10-fold cross validation round. Execution times are relative to the smallest execution time observed in the models benchmarking. For the moment, let us just focus on the first row of each model (i.e., features mode *full*), which corresponds to the performance using all the input features. Whereas MLP and RF models provide very similar detection results, training the MLP model takes much longer than training the RF one, i.e., 3 orders of magnitude longer times. This makes of RFs a very appealing

Table II: Area under the ROC curve and relative execution time on MAWI dataset.

model	features mode	DDoS		mptp-la		ping-flood		netscan-UDP		netscan-TCP (ACK)	
		ROC	relative ET	ROC	relative ET	ROC	relative ET	ROC	relative ET	ROC	relative ET
CART	full	0.922	27.8	0.972	12.3	0.952	20.8	0.972	14.4	0.918	22.3
	FS	0.924	**1.9**	0.944	**1.4**	0.972	**1.7**	0.958	**1.8**	0.938	**2.2**
	top PLCC	0.874	1.6	0.965	1.9	0.956	6.1	0.970	4.9	0.920	6.0
MLP	full	**0.995**	27.4×10^3	**0.998**	27.3×10^3	**0.996**	27.3×10^3	**0.997**	27.3×10^3	**0.997**	27.4×10^3
	FS	0.947	59.5	0.979	76.1	0.993	74.6	0.994	13.8	0.989	72.8
	top PLCC	0.869	25.6	0.993	21.6	0.997	1.4×10^3	0.994	1.1×10^3	0.993	1.2×10^3
Naïve Bayes	full	0.828	29.3	0.952	27.4	0.963	26.5	0.944	26.4	0.929	26.1
	FS	0.863	1.36	0.993	2	0.969	2	0.969	2.1	0.950	2
	top PLCC	0.824	1.1	0.982	3.2	0.981	4.9	0.925	4.3	0.959	4.6
Random Forest	full	**0.979**	10.5	**0.998**	4.6	**0.996**	7.5	**0.995**	7.2	**0.989**	7.5
	FS	0.984	5	0.987	2.6	0.995	3.8	0.995	4	0.990	4.5
	top PLCC	0.942	4.4	0.988	2.4	0.991	5.3	0.996	5.8	0.988	5.7
SVM	full	0.935	37.6	0.982	5.5	0.980	13.5	0.982	11.2	0.968	18.3
	FS	0.803	**3**	0.916	**1**	0.932	**2.8**	0.933	**2**	0.877	**2.2**
	top PLCC	0.755	2.2	0.982	1.1	0.948	3.4	0.938	3.9	0.900	4

solution for the application of Big-DAMA to the detection of network attacks. Next we show that execution performance can still be improved by using less and more relevant input features for the analysis.

C. Improving Execution Time by Feature Selection

While using a large set of input features can normally result in improved performance for some supervised approaches, it is not always the best strategy to follow, as it may negatively impact execution performance. Using more features increments the dimensionality of the feature space, normally introducing undesirable effects such as sparsity and training over-fitting. At the same time, using irrelevant or redundant features may diminish performance in the practice [4]. We show next that by carefully addressing the pre-filtering of input features by standard feature selection techniques, we can dramatically reduce the execution times.

There are different search strategies and evaluation criteria to construct a sub-set of traffic features. We consider correlation-based selection, following two different approaches: (i) plain top, individually linearly correlated-to-the-target feature selection - referred to as *top PLCC*, in which we simply take the mostly linearly correlated features for each attack type; (ii) sub-set search selection - referred to as *FS*, in which we select sub-sets of features that are poorly correlated among each other, but highly correlated to the targets. In this case, we use Best-First search as search strategy. Fig. 3 shows the absolute values of the linear correlation coefficients (PLCC) between features and attacks, separated by attack type. Features are sorted by decreasing correlation coefficient magnitude. A first observation is that features are in general poorly correlated to the attacks, with PLCC values generally below 0.5. Note that less input features are highly correlated to the DDoS class, which justifies the poorer performance obtained for this attack type. For the sake of top-PLCC feature selection, we keep features with a PLCC coefficient value above 0.2, resulting in 11, 29, 51, 45 and 47 features for the DDoS, HTTP flashcrowd, ping flood, UDP and TCP netscans attacks respectively. In the case of FS feature selection, the procedure selects 13, 19, 19, 21 and 19 features respectively. Tab. II shows also the area under the ROC curve and the total execution time for these feature selection approaches. Note that in all cases, there is a significant reduction on the

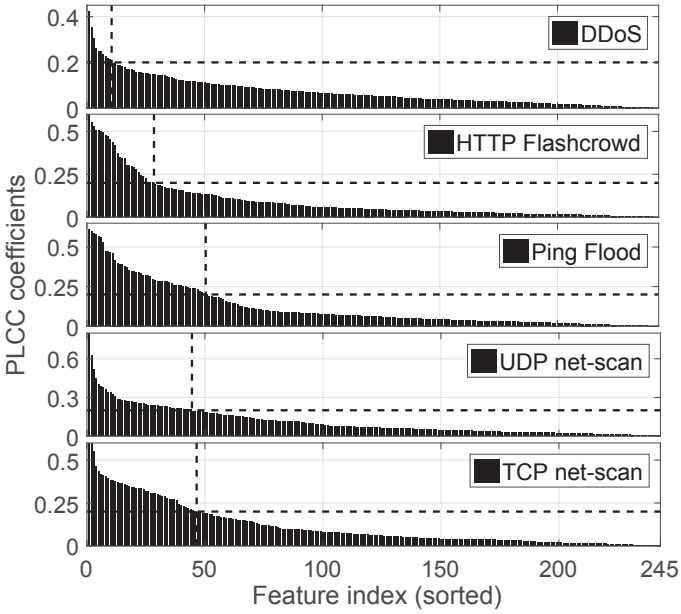


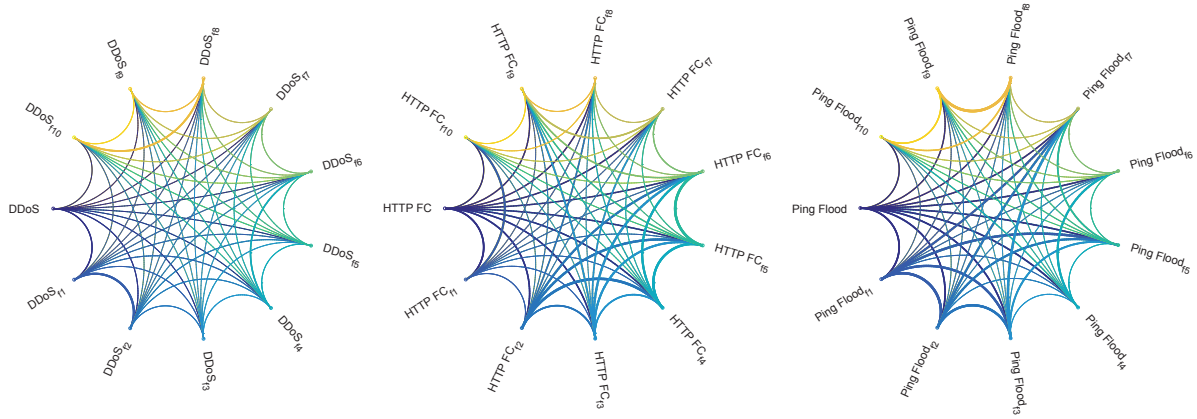
Figure 3: Linear correlation between features and attacks (absolute values). Features are sorted by correlation coefficient magnitude. p-values are below 0.01 for the flagged, top correlated features.

execution times, with an associated reduction on the detection performance. Still, for the two best models, namely MLP and RF, detection results are still highly accurate. Interesting is the case of the CART decision tree model, for which performance even slightly increases for some types of attacks, with a dramatic reduction on the execution times.

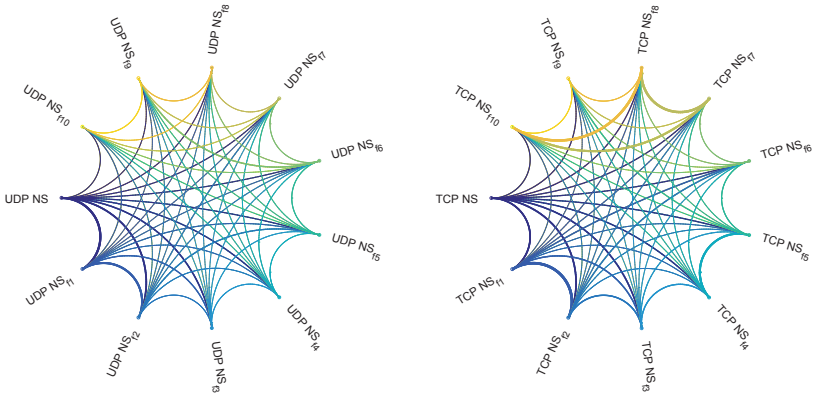
To get a better understanding on which are the best features to detect the studied attacks, Tab. III reports the top-10 correlated features per attack type, and Fig. 4 shows the inter-feature correlations among each set of 10 features, in the form of a circular graph. Features are coherent with the characteristics of each attack type, e.g., having a large number of packets towards a top targeted destination IP and destination port in the case of a DDoS attack. Note that in all cases, features derived from the empirical distributions are present in the top-10 features, suggesting that such types of features, generally not computed in other studies, are highly relevant for the sake of detection of network attacks.

Table III: Top-10 correlated features per attack type.

	DDoS	HTTP flashcrowd	Ping flood	UDP netscan	TCP netscan
f_1	# pkts	% pkts $\rightarrow$ +TCP$_{\text{dst-port}}$	% IPv4 pkts	head p (IP$_{\text{len}}$)	% IPv4 pkts
f_2	% pkts $\rightarrow$ +IP	$\overline{p}$ (TCP$_{\text{dst-port}}$)	% IPv6 pkts	head p (IP$_{\text{len}}$)	% IPv6 pkts
f_3	tail p (TCP$_{\text{src-port}}$)	head p (TCP$_{\text{dst-port}}$)	$\overline{p}$ (IP$_{\text{len}}$)	head p (UDP$_{\text{dst-port}}$)	tail p (TCP$_{\text{src-port}}$)
f_4	tail p (UDP$_{\text{dst-port}}$)	head p (TCP$_{\text{dst-port}}$)	% ICMP pkts	% pkts $\rightarrow$ +IP	head p (TCP$_{\text{win-size}}$)
f_5	tail p (TCP$_{\text{dst-port}}$)	tail p (TCP$_{\text{dst-port}}$)	% pkts $\rightarrow$ +IP	tail p (UDP$_{\text{src-port}}$)	head p (TCP$_{\text{win-size}}$)
f_6	head p (UDP$_{\text{src-port}}$)	tail p (TCP$_{\text{dst-port}}$)	# dst IPs	$\overline{p}$ (IP$_{\text{len}}$)	# TCP$_{\text{dst-ports}}$
f_7	# TCP$_{\text{dst-ports}}$	head p (TCP$_{\text{win-size}}$)	head p (IP$_{\text{len}}$)	% UDP pkts	$\overline{p}$ (TCP$_{\text{dst-port}}$)
f_8	head p (IP$_{\text{TTL}}$)	% pkts $\rightarrow$ +IP	head p (IP$_{\text{TTL}}$)	tail p (UDP$_{\text{dst-port}}$)	tail p (TCP$_{\text{dst-port}}$)
f_9	# src IPs	H (p (TCP$_{\text{dst-port}}$))	head p (IP$_{\text{TTL}}$)	# dst IPs	head p (TCP$_{\text{dst-port}}$)
f_{10}	head p (IP$_{\text{TTL}}$)	tail p (TCP$_{\text{src-port}}$)	# src IPs	# UDP$_{\text{dst-ports}}$	$\hat{p}$ (TCP$_{\text{dst-port}}$)



(a) DDoS (b) HTTP flashcrowd (c) Ping flood



(d) netscan UDP (e) netscan TCP (ACK)

Figure 4: Top-10 feature correlation graphs for the different types of attack.

D. Big-DAMA vs. DBStream vs. Spark

To conclude the paper, we benchmark the performance of Big-DAMA against other big data solutions used for network monitoring. We use the benchmark developed in [8] for the sake of comparing network monitoring systems, but slightly modify it to fit the network security tasks and datasets used in this paper. The benchmark consists of 7 different, inter-dependent analysis tasks or jobs, which are representative of the standard operations performed for network traffic monitoring and analysis. These jobs consists of recursively updated results, with different time-window batch lengths (i.e., processing micro-batches of 1 minute to long batches of 1 hour) and different analysis complexity. The 7 jobs can be briefly described as follows: J_1: every 10 minutes, map source/destination IPs to the underlying Autonomous System, using *team cymru* IP2ASN mapping lists (http://www.team-cymru.org/), and compute aggregated traffic statistics, such as min/max/avg RTT, number of distinct IPs, total number of uploaded/downloaded bytes; J_2: every hour, compute same traffic statistics as for J_1, but aggregating flows by source AS; J_3: every hour, map source IPs to AS numbers, and select the top 10 ASes having the biggest number of IPs; J_4: every hour, aggregate traffic into /24 subnetworks, and select the top 10 /24 subnets with the biggest number of flows; J_5: every minute, compute the total number of flows and the uploaded/downloaded bytes for each source IP; J_6 and J_7 are incremental queries, which take advantage

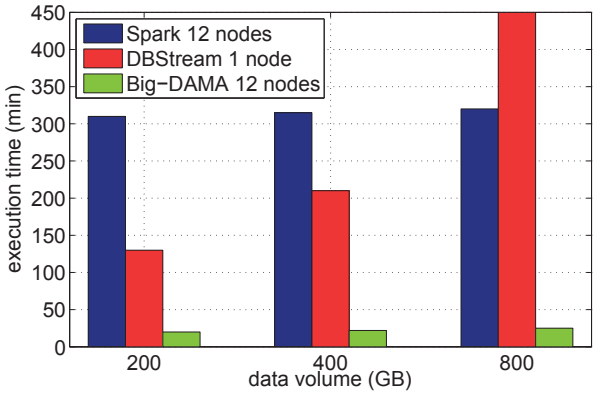


Figure 5: Performance comparison in terms of execution time in processing MAWI data. Big-DAMA can speed up computations by a factor of 10 when compared to a standard Apache Spark cluster.

of the DSW architecture: in J_6, for every minute compute the set of different destination IPs, and use it to update the set of destination IPs which were active during the past hour; finally, in J_7, for every minute compute the number of uploaded/downloaded bytes per source IP, and compute the moving average during the past hour.

We compare Big-DAMA against two other solutions: our previously conceived DBStream system [8], which is a BDAF for network monitoring, and a standard, Spark stand-alone cluster. Both Big-DAMA and Spark are deployed on the 12-node virtualized infrastructure described in Sec. III. DBStream does not support distributed infrastructures, and runs only on a single node. While this makes the comparison rather biased for DBStream, we have shown in [8] that even such a 1-node deployment can outperform a Spark cluster for recursive analysis tasks as those available in the benchmark, in particular jobs J_6 and J_7. Evaluations are done on top of the datasets analyzed in this paper.

Fig. 5 reports the obtained results, considering different volumes of analyzed data: 200 GB, 400 GB and 800 GB of traffic. The first observation is that the comparative results between DBStream and the Spark cluster previously obtained in [8] still hold: when traffic volumes are smaller, DBStream is capable to outperform a 12-node Spark cluster; this is mainly due to the underlying characteristics of both systems, especially regarding the recursive jobs; DBStream is particularly tailored to handle such type of analytics, whereas Spark is meant for pure batch processing. Note however that the Spark results are almost the same for the three data volumes, clearly showing that the bottleneck is not on the cluster capacity, but rather on the characteristics of Spark (data import times are not considered in the benchmark). Big-DAMA is capable to reduce the plain Spark cluster execution time by an order of magnitude, taking about 20 minutes instead of the more than 300 minutes taken by Spark. Contrary to DBStream, Big-DAMA uses the power of the full 12-node cluster, and thus can scale out linearly as compared to a single node. The combined usage of Spark streaming for handling small batches and the recursive nature of the DSW architecture to merge recursive results on the Cassandra DB offer a high performance data analytics system.

V. Concluding Remarks

We have presented Big-DAMA, a big data analytics framework specially tailored for network monitoring applications. Using off-the-shelf big data storage and processing engines, Big-DAMA is capable of analyzing and storing big amounts of both structured and unstructured heterogeneous data sources, with both stream and batch processing capabilities. We have shown the types of ML-based algorithms implemented in Big-DAMA for network security, using off-the-shelf ML libraries. By applying Big-DAMA to the detection of different types of network attacks on top of real network measurements collected at the WIDE backbone network, we have also explored novel features to better and faster detecting common network attacks. The analysis of feature selection techniques also showed that it is possible to further reduce execution times by keeping only the most relevant and correlated-to-the-target features. Finally, we have shown that Big-DAMA can speed up computations by an order of magnitude when compared to a standard Apache Spark cluster, and can largely outperform our previous DBStream solution for network traffic monitoring and analysis.

References

[1] R. Fontugne et al., "MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking", *CoNEXT*, 2010.

[2] V. Chandola et al., "Anomaly Detection: a Survey", *Com. Sur.*, vol. 41 (3), 2009.

[3] T. Nguyen et al., "A Survey of Techniques for Internet Traffic Classification using Machine Learning", *IEEE Comm, Surv. & Tut.*, vol. 10 (4), pp. 56-76, 2008.

[4] P. Casas et al., "Unsupervised Network Intrusion Detection Systems: Detecting the Unknown without Knowledge", *Com. Comm.*, vol. 35 (7), 2011.

[5] P. Fiadino et al., "RCATool - A Framework for Detecting and Diagnosing Anomalies in Cellular Networks", *ITC 27*, 2015.

[6] P. Casas et al., "Big-DAMA: Big Data Analytics for Network Traffic Monitoring and Analysis", *SIGCOMM LANCOMM workshop*, 2016.

[7] P. Casas et al., "(Semi)-Supervised Machine Learning Approaches for Network Security in High-Dimensional Network Data", *CCS*, 2016.

[8] A. Baer et al., "Large-Scale Network Traffic Monitoring with DBStream, a System for Rolling Big Data Analysis," *IEEE Big Data*, 2014.

[9] L. Golab et al., "Stream Warehousing with DataDepot," *SIGMOD*, 2009.

[10] M. Stonebraker, "SQL Databases vs. noSQL Databases," *Comm. of the ACM*, vol. 53(4), pp. 10-11, 2010.

[11] C. Cranor et al., "Gigascope: A Stream Database for Network Applications," *SIGMOD*, 2003.

[12] D. Abadi et al., "Aurora: A New Model and Architecture for Data Stream Management," *The VLDB Journal*, 12(2), pp. 1020-1039, 2003.

[13] J. Dean et al., "MapReduce: Simplified Data Processing on Large Clusters," *Comm. of the ACM*, 51(1), pp. 107-113, 2008.

[14] T. White, "Hadoop: the Definitive Guide," *O'Reilly Media, Inc.*, 2009.

[15] M. Zaharia et al., "Spark: Cluster Computing with Working Sets," *HotCloud*'10.

[16] M. Zaharia, et al., "Discretized Streams: An Efficient and Fault-tolerant Model for Stream Processing on Large Clusters," *HotCloud*, 2012.

[17] P. Bhatotia et al., "Indoop: Mapreduce for Incremental Computations," *SoCC*'11.

[18] W. Lam et al., "Muppet: Mapreduce-style processing of fast data," *Proc. VLDB Endow.*, vol. 5(12), pp.1814-1825, 2012.

[19] B. Li et al., "Scalla: A platform for scalable one-pass analytics using mapreduce," *ACM Trans. Database Syst.* 37(4), pp. 27-43, 2012.

[20] R. Fontugne et al., "Hashdoop: A MapReduce Framework for Network Anomaly Detection," *IEEE INFOCOM Workshops*, 2014.

[21] Y. Lee et al., "Toward scalable internet traffic measurement and analysis with Hadoop," in *SIGCOMM Comput. Commun. Rev.*, 43(1), pp. 5-13, 2012.

[22] J. Liu et al., "Monitoring and analyzing big traffic data of a large-scale cellular network with Hadoop," *IEEE Network*, 28(4), pp. 32-39, 2014.

[23] M. Wullink et al., "ENTRADA: a High-Performance Network Traffic Data Streaming Warehouse," *IEEE/IFIP NOMS*, 2016.

[24] S. Melnik et al., "Dremel: Interactive Analysis of Web-scale Datasets," *Proc. VLDB Endow.*, 3, pp. 330-339, 2010.

[25] T. Akidau et al., "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," in *VLDB 41*, 2015.