Industrial Engineering Journal
ISSN: 0970-2555
Volume: 52, Issue 3, March 2023

NETWORK SECURITY AND ANOMALY DETECTION WITH BIG-DAMA, A BIG DATA ANALYTICS FRAMEWORK

[1]Dr B Ravi Prasad, Professor, CSE Department, Marri Laxman Reddy Institute of Technology and Management, Hyderabad, rprasad.boddu@gmail.com
[2]Dr.Aluri Brahmareddy, Associate professor, CSE Department, Marri Laxman Reddy Institute of Technology and Management, Hyderabad brahmareddy475@gmail.com

ABSTRACT:
Network traffic monitoring and analysis systems have become increasingly difficult to develop as both the Internet's complexity and network traffic volume have skyrocketed in recent years. Mechanisms for on-line analysis of thousands of events per second and effective ways for off-line analysis of huge historical data are essential for critical NTMA applications like the detection of network assaults and abnormalities. The high dimensionality of network data provided by modern network monitoring systems allows for the widespread application of machine learning approaches to enhance the detection and classification of network attacks and anomalies, but this higher dimensionality comes at the cost of an increased data processing overhead. In this study, we introduce Big-DAMA, a big data analytics framework designed specifically for NTMA use cases. Big-DAMA is a versatile BDAF that can handle both stream and batch processing to evaluate and store massive volumes of data from a variety of heterogeneous sources. Big-DAMA follows the Data Stream Warehouse paradigm by utilizing commercially available big data storage and processing engines to provide both stream data processing and batch processing capabilities. This is accomplished by decomposing these engines into three distinct parts: stream, batch, and query. Several techniques for anomaly detection and network security are implemented in Big-DAMA by making use of supervised and unsupervised machine learning models and pre-existing ML libraries. We use Big-DAMA to compare the performance of many supervised ML models in detecting abnormalities and assaults in networks. Measurements taken from the WIDE backbone network are used in conjunction with the popular MAWILab dataset to classify attacks and then put through a series of tests to determine performance. By employing hardware virtualization technology, Big-DAMA may be readily deployed in cloud settings, where it can speed up calculations by a factor of 10. This is in comparison to a regular Apache Spark cluster.

Keywords: Big-Data, Network Traffic Monitoring and Analysis, Network Attacks, Machine Learning, High-Dimensional Data & MAWILab.

I. INTRODUCTION
Today, Network Traffic Monitoring and Analysis plays a crucial role in elucidating the functioning of complex and huge networks, especially when issues arise. The analysis and processing of very large amounts of rapidly changing, heterogeneous network monitoring data is a major challenge for large NTMA applications. The data collected for network monitoring typically arrives in the form of high-velocity streams that must be continuously processed and analyzed. Many methods have been proposed throughout the years to acquire extensive data from live networks. In order to evaluate and draw conclusions from such voluminous data, a data processing system with sufficient flexibility is required. Novel solutions for large data processing have been more widely used due to their rapid development. However, the platforms that have

been conceived are vastly different from one another, each with distinct requirements and aims. Each big data expert, therefore, must fumble their way through the myriad of options. The same is true for big data analytics using ML-based techniques; despite the proliferation of ML libraries for big data platforms, there is a significant chasm between the two fields.

In this paper, we present Big-DAMA, a Big Data Analytics Framework for network traffic monitoring applications. Based on the foundation of an earlier system called DBStream, we've developed an early version of the Big-DAMA BDAF prototype. Big-DAMA is a versatile BDAF that can handle both stream and batch processing to evaluate and store massive volumes of data from a variety of heterogeneous sources. Big-DAMA uses both supervised and unsupervised machine learning models to implement multiple data analytics algorithms for network security and anomaly detection. Standard machine learning libraries are used to implement these models. At now, Big-DAMA is being used in a test capacity as a cluster built atop virtual hardware technologies. We apply the Big-DAMA BDAF to real datasets collected from the WIDE backbone operational network to demonstrate its use in an operational NTMA application, specifically the detection of network attacks and traffic anomalies. Network assaults and traffic irregularities are difficult to detect automatically since they are always changing. Since both novel attacks and variations on existing ones are constantly appearing in the wild, it is challenging to clearly and continually describe the set of probable anomalies that may develop, especially in the case of network attacks. This means that ML-based models show promise for capturing the underlying properties of such unexpected occurrences, and that a generic anomaly detection system should be able to identify a broad variety of anomalies with various structures. This study extends our prior preliminary work on using big data analytics and machine learning to improve network security.
The reminder of the paper is structured as follows. Section II presents an overview on the related work. In Section III we 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 so 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

III. The big DAMA Framework

Big-primary DAMA's use case is in network analysis and data storage for massive volumes of monitoring information. Following a normal lambda design, Big-DAMA decomposes different frameworks for stream, batch, and query in order to provide support for both stream data processing and batch processing. Lambda architecture is an approach to data processing that combines batch and stream processing to deal with large data sets. The idea behind this design is to strike a good balance between latency, throughput, and fault tolerance by using batch processing to offer complete and accurate views of batch data and real-time stream processing to enable on-the-fly data analysis.
For stream-based analysis, Big-DAMA employs Apache Spark streaming; for batch analysis, Apache Spark; and for query and storage, Apache Cassandra. Cassandra's fault-tolerance and performance make it preferable to other HDFS and Hadoop-based DBs like HBase and

Hive. While HDFS including HBase and Hive has a single point-of-failure represented by the HDFS name nodes, Cassandra is completely distributed and has no single point of failure since every node has the same function. While HDFS allows for many name nodes to be established, failure tolerance may still be a problem due to the roles-split. While HDFS has a more traditional data warehousing approach, Cassandra was designed from the ground up to handle real-time transactional data. When compared to HDFS DBs like HBase, Cassandra performs much better in terms of performance and latency. In comparison to other NoSQL databases like HBase, MongoDB, and Couch base, newer benchmarks1 show that Cassandra excels at the kinds of tasks typically performed by real-world applications. When it comes to scalability, Cassandra excels, offering linear scalability without sacrificing processing performance. And lastly, as a NoSQL system, it facilitates the storage and management of data from a wide variety of sources, including unstructured data.
Figure 1 depicts Big-overall DAMA's architectural plan. The Big-DAMA BDAF, which draws inspiration from DBStream, is organized according to a DSW paradigm and provides the opportunity to integrate real-time processing with massive storage and analytics. This paradigm allows for simultaneous on-line and off-line processing needs to be met by a single infrastructure.

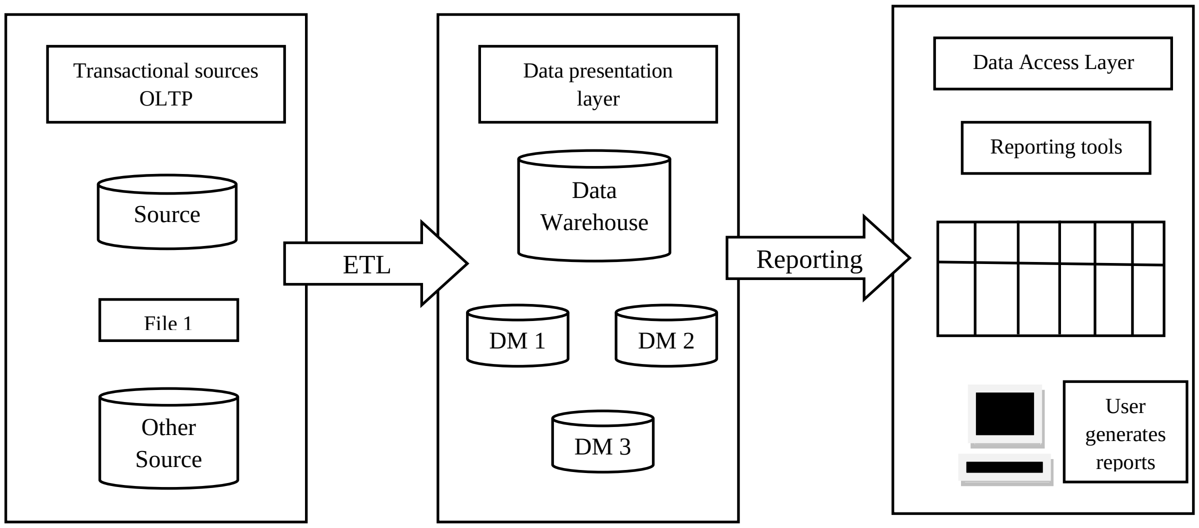

Figure 1: Big- DAMA is built on top of the Hadoop ecosystem and relies on a data warehouse for its infrastructure.

Within the Big-DAMA BDAF, we have developed different algorithms for network security and anomaly detection using supervised and unsupervised machine learning models. Too far, these models have been built on top of the Big-DAMA batch-processing branch, using commercially available Spark ML machine learning libraries.
We are currently investigating the possibility of adopting Apache Beam, an advanced unified programming model, which follows the recently introduced data flow model and simplifies the implementation of both batch and streaming data processing jobs that can run on the base Spark execution engine used by Big-DAMA.

Currently, Big-DAMA is running on top of a virtualized data cluster with 12 virtual nodes, each with 150 GB of RAM and 30 TB of storage space, all linked together via Open switch. The physical infrastructure consists of 3 physical server nodes, each equipped with 2 sinter Xeon(R) E5-2630 v2-2.60GHz CPUs (24 cores total) \sand with a total capacity of 64 GB of memory. VMs running on the Linux kernel do the virtualization, while the Proxmox Virtual Environment (https://www.proxmox.com/) is used to manage and orchestrate the VMs. Big data frameworks are partially managed by a Cloudera, Hadoop ecosystem installation \s(https://www.cloudera.com/), utilizing distribution CDH 5.10 and Cloudera Manager (with Spark 2).

IV. Detecting network attacks with big DAMA

We use the widely-used MAWILab dataset for attacks labelling to analyze actual network traffic measurements taken from the WIDE backbone network to demonstrate Big-DAMA in action. Since 2001, MAWILab has been collecting daily 15-minute network traffic traces on a backbone connection between Japan and the US and making them publicly available. Using this data as a starting point, the MAWILab project employs a hybrid of four time-tested anomaly detectors to perform partial labeling of the traffic.
Starting with a huge number of traffic characteristics retrieved from the packet traces as input, we compare the execution durations and detection accuracy of five different supervised ML models. Finally, we conduct an in-depth analysis of how well the various extracted features perform in detecting the examined assaults, and we compare and contrast various feature selection strategies for retaining the most important characteristics and improving execution speeds.

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, 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, 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, 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, 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, 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, TCPF
	win p{1,2,5,...95,97,99}	percentiles

Finally, we compare Big-performance DAMA's to that of existing big data platforms, demonstrating that the proposed system not only achieves excellent accuracy but also has the potential to surpass other platforms of its kind in terms of execution times.
Field Feature Description.

A. Data Description & ML Models

This study analyses data from two months' worth of packet traces, which were gathered at the end of 2015. From the assaults and anomalies that have been identified, we zero in on those that are simultaneously picked up by all four MAWILab detectors, with a special emphasis on the ones that have been deemed unusual by MAWILab.
We focus on five distinct assaults and anomalies: I distributed denial-of-service attacks (DDoS), (2) HTTP flash crowds (mptp-la), (3) Flooding attacks (Ping flood), and (4) UDP and (5) TCP-ACK probing traffic as part of a distributed network scan (netscan). To do this, we train five distinct ML models, each of which is capable of detecting one of the five aforementioned attack types, and then we run all of these detectors in parallel on top of the data. As a consequence, each method of detection is able to not only identify the existence of an attack but also to categories its kind.
For the detectors' conception, we choose among five fully-supervised models: CART Decision Trees (CART), Random Forest (RF), Support Vector Machines (SVM), Na ve Bayed (NB), and Neural Networks (MLP).

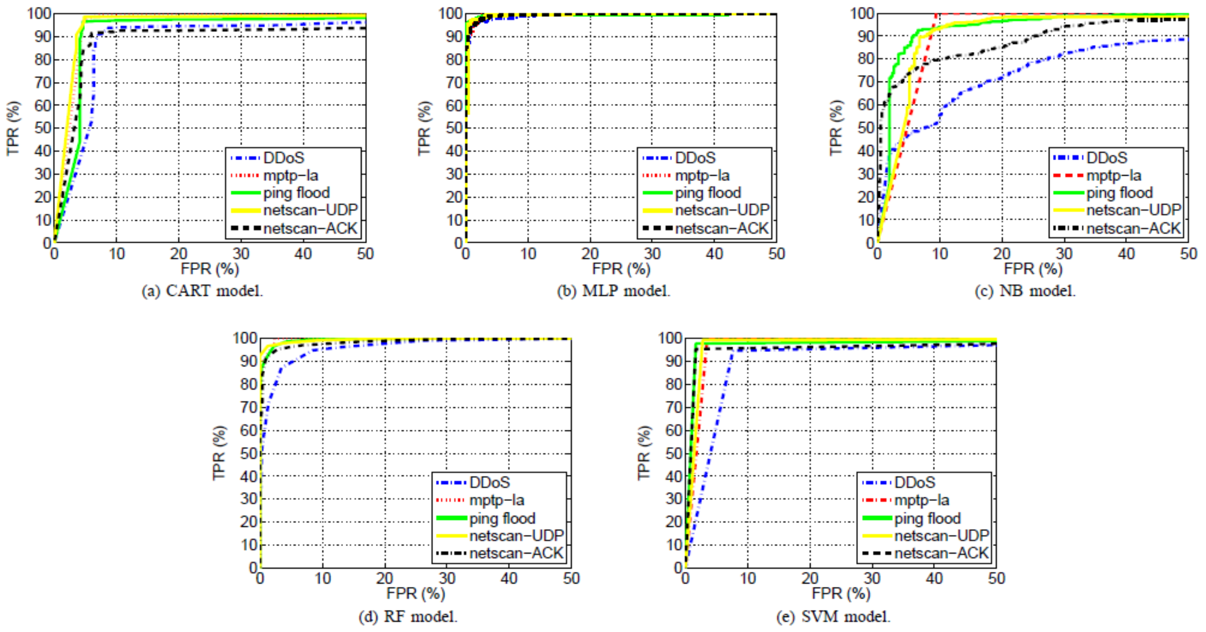

Figure 2: Detection performance per type of attack and ML-based approach.

We chose these detectors because they have shown to be effective in the context of earlier work on anomaly detection and traffic categorization. To fine-tune these ML-based algorithms and conduct the analyses, we make use of the Spark ML Machine-Learning libraries. For more detail on the algorithms and their setup options, we refer the curious reader to the survey and the Spark ML documentation. Our assessment takes into account a time-slotted breakdown of anomaly types and their respective detection efficiencies. To do this, we calculate a collection of features defining the traffic in each one-second time slot extracted from the traffic traces. Furthermore, a label li, consisting of a binary vector li 2 R51, is applied to each slot I where each position denotes whether or not slot I has an anomaly of type j = 1..5.
Conventional packet measurements such as traffic throughput, packet sizes, IP addresses and ports, transport protocols, flags, etc. are used to generate a large number (n) of features describing a time slot, which are then used to better detect attacks and anomalies. There are a total of n = 245 characteristics calculated for each of the I = 1..m time slots, and they are all listed in Tab. I. Please take into account that we not only use conventional characteristics like the minimum, mean, and maximum of some of the input data, but also take into account the empirical distribution of some of them, sampling the empirical distribution at a variety of percentiles. Since the whole distribution is considered, the information used as input is much enhanced. We also calculate the empirical entropy H (.), which reflects the dispersion of the samples in the given time interval, for each of these distributions.

B. Detection Performance with Full Features

By calculating the True and False Positive Rates for each attack type using the entire set of 245 characteristics as input, we evaluate the detection/classification capabilities of the five supervised techniques. The ROC curves for each detector and each suggested attack class are shown in Fig.

Industrial Engineering Journal
ISSN: 0970-2555
Volume: 52, Issue 3, March 2023

2. All given findings are in agreement with 10-fold cross validation, which helps prevent over-fitting. The chosen supervised detectors' comparison findings are shown in Fig. 2. There is no other method, other than the NB model, that yields more precise findings across all five assault types. DDoS assaults have significantly worse detection performance overall. Both the MLP and RF models function optimally, correctly identifying around 80% of assaults.
In Table II, we provide the area under the ROC curve and the total execution time for the whole 10-fold cross validation cycle for each model. Timings are reported as a percentage of the quickest time to completion during benchmarking of the models.
For the time being, we will just look at the first row of each model (i.e., features mode full), which displays the results when all the input characteristics are used. Despite the fact that MLP and RF models provide almost identical detection results, training the former takes much longer (by a factor of three) than training the latter. Because of this, RFs are an attractive option for using Big-DAMA to identify network threats. Our next demonstration demonstrates how fewer but more relevant input characteristics might further boost execution speed.

C. Improving Execution Time by Feature Selection

While employing a high number of input features tends to boost the performance of certain supervised techniques, this is not always the ideal strategy to pursue since doing so might have a detrimental effect on execution speed. Increasing the number of features used raises the feature space's dimensionality, which may cause problems like sparsely and training over-fitting. In the same vein, features that aren't necessary or that are redundant may have a negative impact on efficiency. Next, we demonstrate how much execution times may be reduced by paying close attention to the pre-filtering of input characteristics using common feature selection approaches.
In order to build a more specific list of traffic characteristics, several search algorithms and assessment criteria may be used. For our purposes, we divide correlation-based selection into two camps: I a sub-set search selection, or FS, in which we take the sub-sets of features that are poorly correlated among each other but highly correlated to the targets; and (ii) a top PLCC feature selection, in which we simply take the most linearly correlated features for each attack type. To do this, we use the Best-First search method. Absolute values of the partial linear correlation coefficients (PLCC) between characteristics and assaults, broken down by attack type, are shown in Fig. 3.

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	0.926	27.9	0.975	12.4	0.953	20.9	0.973	14.5	0.919	22.4
	FS	0.926	1.9	0.945	1.4	0.973	1.7	0.959	1.8	0.939	2.2
	top PLCC	0.895	1.7	0.966	1.8	0.957	6.2	0.972	5	0.922	6.2
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	0.948	59.7	0.979	76.3	0.995	74.7	0.995	13.9	0.989	72.9
	top PLCC	0.871	25.7	0.995	21.7	0.998	1.4×10^3	0.995	1.1×10^3	0.994	1.2×10^3
Naive	full	0.831	29.5	0.953	27.5	0.967	26.6	0.945	26.5	0.929	262

Bayes	FS	0.865	1.37	0.995	2	0.969	3	0.969	2.3	0.952	2
	top PLCC	0.826	1.2	0.983	3.3	0.983	4.9	0.936	4.5	0.959	4.7
Random Forest	full	0.979	10.6	0.998	4.7	0.996	7.5	0.995	7.3	0.989	7.6
	FS	0.985	6	0.988	2.7	0.996	3.9	0.996	4	0.992	4.6
	top PLCC	0.943	405	0.989	2.5	0.992	5.6	0.997	5.9	0.989	5.8
SVM	full	0.936	37.7	0.983	5.6	0.982	13.6	0.983	11.3	0.969	18.5
	FS	0.805	3	0.917	1	0.934	2.8	0.934	2	0.879	2.2
	top PLCC	0.756	2.2	0.983	1.2	0.959	3.5	0.939	3.9	0.901	5

The magnitude of the correlation coefficient is used to rank the features. First, PLCC values are often below 0.5, indicating that attributes are not strongly connected with the assaults. It is important to keep in mind that the lower performance achieved for the DDoS attack type might be explained by the fact that fewer input attributes are significantly connected to the DDoS class. Keeping only features with a PLCC coefficient value greater than 0.2 allows us to select the top-PLCC features for each type of attack, yielding 11, 29, 51, 45, and 47 features for DDoS, HTTP flash crowd, ping flood, UDP, and TCP nets can, respectively. Choose 13 features, 19 features, 21 features, and 19 features, respectively, for FS. The execution time and area under the receiver operating characteristic curve for each of these feature selection methods are shown in Tab. II. Note that in all cases, there is a significant reduction on the execution times, with an associated reduction on the detection performance. Still, for the two best models, namely MLP and \sRF, detection results remain highly accurate. Interesting is the scale of the CART decision tree model, for which performance seven 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- \s10 correlated features per attack type, and Fig. 4 shows the \sinter-feature correlations among each set of 10 features, in form of a circular graph. Features are coherent with the characteristics of each attack type, e.g., having large number \sof packets towards a top targeted destination IP and destination \sport in the case of a DDoS attack.

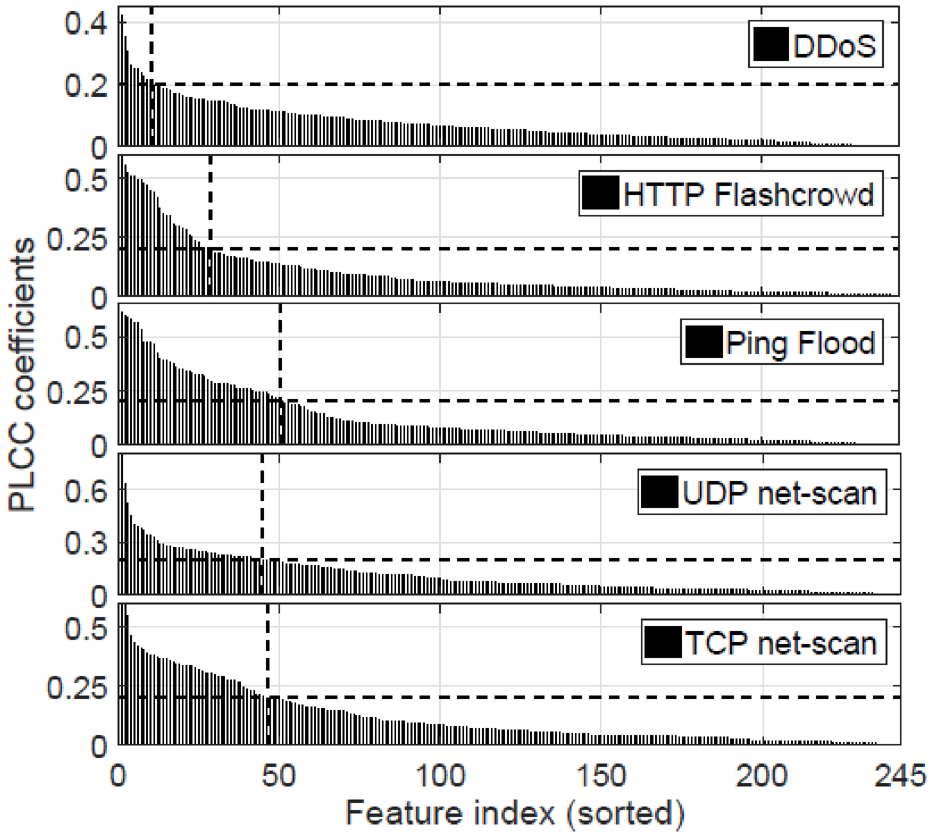

Figure 3: Linear correlation between features and attacks (absolute values). Features are sorted by correlation coefficient magnitude. P-values are below 0.01 for the flagged, top correlated features.

Note that in all cases, features derived from the empirical distributions are present \sin 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.

D. Big-DAMA vs. DBStream vs. Spark

In the last section of the study, we compare Big-DAMA to other big data systems for network monitoring. To compare different network monitoring systems, we adapt the benchmark for use with our own network security tasks and datasets. The benchmark comprises of seven distinct analysis tasks or jobs that are all interconnected in some way and are meant to be indicative of typical tasks in network traffic analysis. A variety of analytical complexities and time-window batch lengths (from processing micro-batches of 1 minute to lengthy batches of 1 hour) make up these projects.

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

	DDoS	HTTP flash crowd	Ping flood	UDP netscan	TCP netscan
f1	# pkts	% pkts! +TCPdst-port	% IPv4 pkts	head p (IPlen)	% IPv4 pkts
f2	% pkts! +IP	p TCPdst-port	% IPv6 pkts	head p (IPlen)	% IPv6 pkts

Industrial Engineering Journal
ISSN: 0970-2555
Volume: 52, Issue 3, March 2023

f3	tail p (TCPsrc-port)	head p TCPdst-port	p (IPlen)	head p UDPdst-port	tail p (TCPsrc-port)
f4	tail p UDPdst-port	head p TCPdst-port	% ICMP pkts	% pkts! +IP	head p (TCPwin-size)
f5	tail p TCPdst-por	tail p TCPdst-port	% pkts! +IP	tail p (UDPsrc-port)	head p (TCPwin-size)
f6	head p (UDPsrc-port)	tail p TCPdst-port	# dst IPs	p (IPlen)	# TCPdst-ports
f7	# TCPdst-ports	head p (TCPwin-size)	head p (IPlen)	% UDP pkts	p TCPdst-port
f8	head p (IPTTL)	% pkts! +IP	head p (IPTTL)	tail p UDPdst-port	tail p TCPdst-port
f9	# src IPs	H p TCPdst-port	head p (IPTTL)	# dst IPs	head p TCPdst-port
f10	head p (IPTTL)	tail p (TCPsrc-port)	# src IPs	# UDPdst-ports	p TCPdst-port

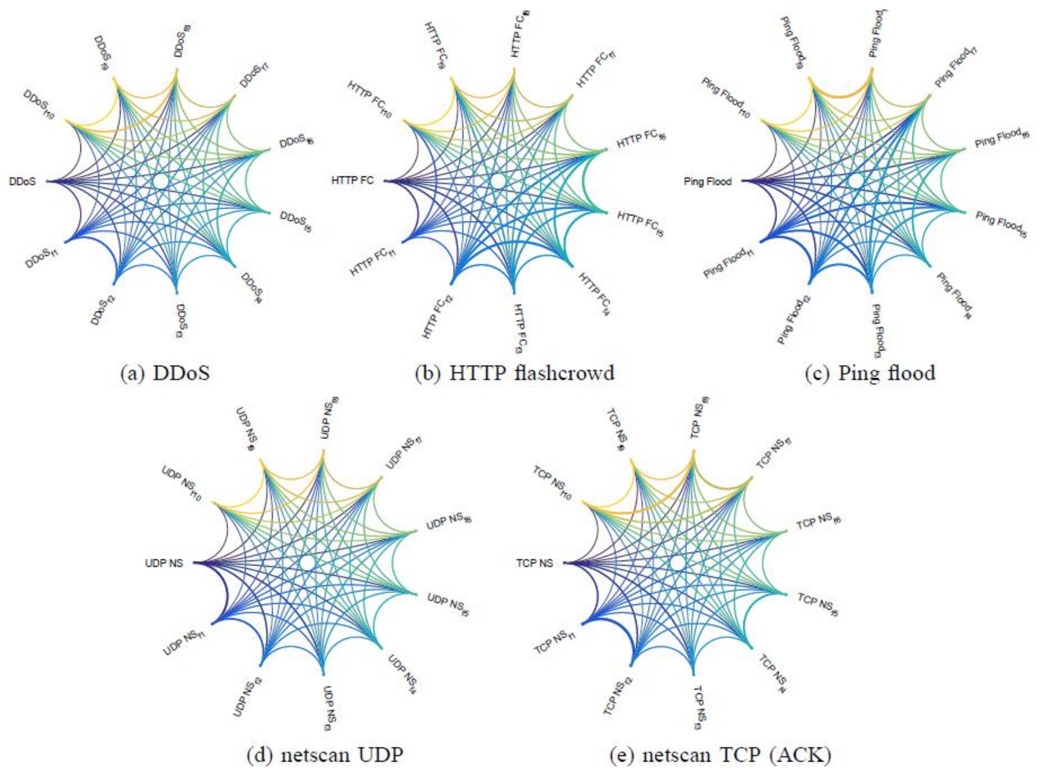

Figure 4: Top-10 feature correlation graphs for the different types of attack.

The seven occupations are summed up as follows: J1: every 10 minutes, use team Cyrus IP2ASN mapping lists (http://www.team-cymru.org/) to map source/destination IPs to the underlying Autonomous System, and then calculate aggregate traffic data like min/max/avg RTT, number of different IPs, and total quantity of uploaded/downloaded bytes; J2: each hour, aggregate flows by

source AS to provide the same traffic data as J1; J3: every hour, choose the 10 autonomous systems with the highest number of IP addresses by mapping source IP addresses to AS numbers; J4: once each hour, choose the top 10 /24 subnets based on the quantity of flows they generate. J5: tally the number of flows and the bytes uploaded/downloaded for each source IP per minute. This is because J6 and J7 are incremental queries that take use of the DSW design: Update the list of active destination IPs from the last hour by computing a new set of destination IPs every minute in J6. Lastly, in J7, you should calculate the moving average of the bytes uploaded/downloaded per source IP per minute over the last hour.
We evaluate Big-DAMA in relation to a standalone Spark cluster and our previously developed DBStream system, a BDAF for network monitoring. The virtualized 12-node environment described in Section III is home to both Big-DAMA and Spark. Unfortunately, DBStream can only be executed on a single node and does not function in a distributed environment.

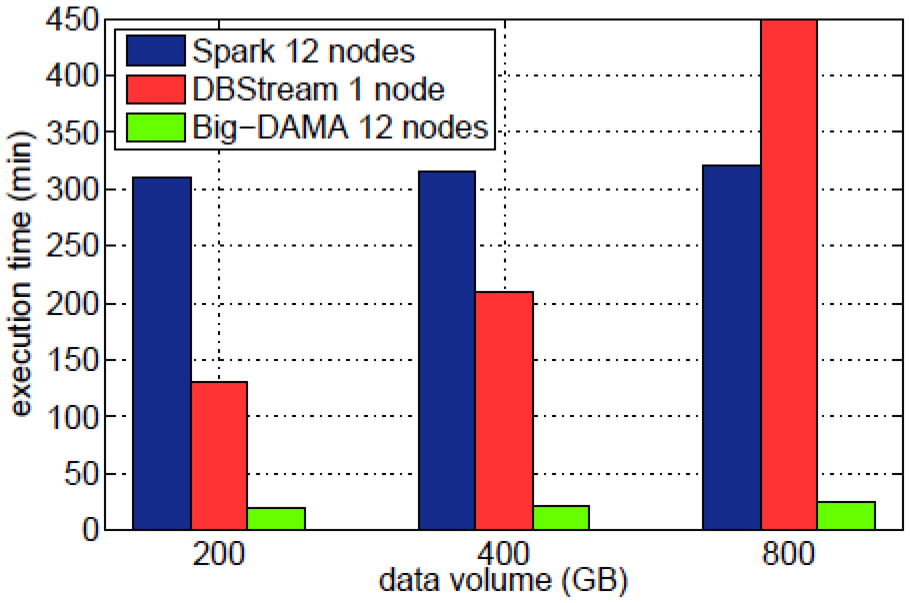

Figure 5: Performance comparison in terms of execution time in processing MAWI data. Big-DAMA can speed up computations by a factor of 10 when compared to a standard Apache Spark cluster.

Even while this skews the comparison in favor of DBStream, we have shown in that a single-node deployment may still beat a Spark cluster on recursive analytical workloads like those seen in the test, especially jobs J6 and J7. Datasets similar to those used in this article are used for further evaluations.
The findings are shown in Fig. 5 for three distinct data sizes (200 GB, 400 GB, and 800 GB) that were studied. The first thing to notice is that the results from the comparison between DBStream and the Spark cluster obtained in still hold: when traffic volumes are low, DBStream is able to outperform a 12-node Spark cluster. This is primarily due to the underlying characteristics of both systems, especially regarding the recursive jobs; DBStream is specifically tailored to handle such analytics, whereas Spark is meant for pure batch processing. It's important to keep in mind, however, that the Spark results are almost same across all three data volumes, indicating that the

bottleneck is not in the cluster capacity but rather in the features of Spark (data import delays are not taken into account in the benchmark). Big-DAMA may lower the execution time of a Spark cluster by an order of magnitude, from more than 300 minutes down to roughly 20. Big- DAMA, in contrast to DBStream, utilizes the whole 12-node cluster, allowing for linear scalability beyond a single node. A high-performance data analytics solution is provided by combining Spark streaming for managing tiny batches with the recursive nature of the DSW architecture for merging recursive findings on the Cassandra DB.

V. Concluding remarks

The findings are shown in Fig. 5 for three distinct data sizes (200 GB, 400 GB, and 800 GB) that were studied. The first thing to notice is that the results from the comparison between DBStream and the Spark cluster obtained in still hold: when traffic volumes are low, DBStream is able to outperform a 12-node Spark cluster. This is primarily due to the underlying characteristics of both systems, especially regarding the recursive jobs; DBStream is specifically tailored to handle such analytics, whereas Spark is meant for pure batch processing. It's important to keep in mind, however, that the Spark results are almost same across all three data volumes, indicating that the bottleneck is not in the cluster capacity but rather in the features of Spark (data import delays are not taken into account in the benchmark). Big-DAMA may lower the execution time of a Spark cluster by an order of magnitude, from more than 300 minutes down to roughly 20. Big- DAMA, in contrast to DBStream, utilizes the whole 12-node cluster, allowing for linear scalability beyond a single node. A high-performance data analytics solution is provided by combining Spark streaming for managing tiny batches with the recursive nature of the DSW architecture for merging recursive findings on the Cassandra DB.

REFERENCES