A Dataflow System for Anomaly Detection and Analysis

Andrei Bara, Xinyu Niu, Wayne Luk
Dept. of Computing, Faculty of Engineering, Imperial College London, UK
{andrei.bara10, niu.xinyu10, w.luk}@imperial.ac.uk

***Abstract*—This paper proposes DeADA, a dataflow architecture incorporating an automated, unsupervised and online learning algorithm. Compared with 24 core software implementations, DeADA achieves up to 6.17 times lower data drop rate and 10.7 times higher power efficiency. More importantly, experimental results for the Heartbleed case study suggest that DeADA is capable of detecting unknown attacks under network speeds of at least 18Mbps, a feature which is essential for modern network intrusion detection.**

I. Introduction

The recent Heartbleed exploit and Bash vulnerability have brought the importance of network security back to the public's attention. One area concerned with network security is network intrusion detection (NID). Various systems have been proposed, from simpler techniques like rule based intrusion detection which can perform *live* intrusion detection, but which has difficulties in handling more complex scenarios, to machine learning algorithms which are smarter, but significantly slower. Both methods have a reduced ability when dealing with *unknown* attack patterns.

We identify two main issues. Firstly, reduced processing speed of the machine learning solutions results in low throughput of the analysis step, and thus an decrease in the probability of discovering anomalies in big data/network data [1]. Secondly, modern systems are structured around big data which is quite often subjected to concept drift [2]. The lack of a scalable and automated solution which can handle concept drift causes the classification accuracy to degrade over time.

Hence, we highlight the following contributions of our paper:

- *DeADA*: a scalable dataflow architecture which acts as an end-to-end system for anomaly detection and analysis, focusing on removing the bottleneck introduced by the decision function of the One-Class Support Vector (OCSVM).
- We analyze the impact of an increased processing rate on the accuracy of the anomaly detection algorithm by using the Heartbleed attack as an example.

II. Anomaly detection

A. One Class Support Vector Machines

One-Class Support Vector Machines (SVM) are a recent addition to the field of machine learning algorithms which build on top of the classical SVMs, and deal with *identifying* whether new data are of the same class as the training data, thus becoming an attractive candidate for anomaly detection techniques. There are two main models used for describing the OCSVMs: one developed by Schölkopf et al. [3], the other one by Tax and Duin [4]. For this paper we will be using Schölkopf's version.

The idea behind Schölkopf's algorithm is to create a function f which maps most of data in some *small* region $+1$ and the rest to -1. During the *offline* phase the OCSVM considers the *origin* point to be the only negative example in the data set and tries to find a separating hyperplane, while maximizing the margin between the data points and the origin [3]. In Figure 1 we can see the learned frontiers of a set of two dimensional data points. These two regions are the result of mapping a higher dimension hyperplane generated using an Radial Basis Function (RBF) kernel back to a two dimensional representation. Although all of the points are part of the same class (i.e. *normal*) only the ones inside the frontier will be classified as normal, the rest being abnormal.

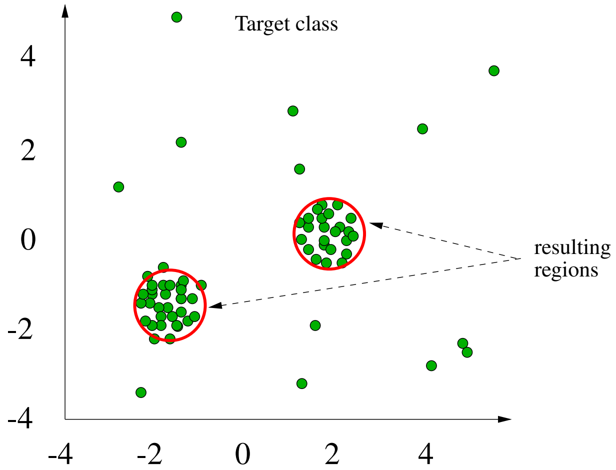

Fig. 1: Learned frontier for One-class SVM.

The decision function of OCSVM is $f(z)$ as shown in Equation 1. λ_i are the coefficients of the support vectors and will be 0 if the vectors are not support. $K(x_i, z)$ is the kernel function, z is the new data vector and ρ is a constant resulted from the OCSVM model generation phase. For this paper we will use a slightly modified version of Equation 1 which has the sgn function removed and call it $OCSVM_{partial}$. This will give us more control in judging *how much* of an anomaly a certain data point is.

$$f(z) = \mathrm{sgn}(\sum_{i}^{n} \lambda_i K(x_i, z) - \rho)) \tag{1}$$

There is research on accelerating the training phase of Support Vector Machines with FPGA. Of particular interest is the work done by Papadonikolakis et al. [5] who propose a geometrical approach to training a classifier which can exploit the heterogeneous capabilities of the FPGA. However, the focus of our paper is in accelerating the classification phase and not on the training phase. In addition, to the best of our knowledge, building a dataflow architecture around One-Class Support Vectors has not been done before. Future research could build on top of the work presented in [5] to speed-up the OCSVM training.

B. Concept drift in anomaly detection

From a technical perspective we can view concept drift as in Figure 2. Here, we trained an initial classifier on $\mathcal{D}_1$ data set resulting in the separating hyperplane shown as a solid line, whereas $\mathcal{D}_2$ and $\mathcal{D}_3$ show two ways in which the change of the hyperplane results in *concept drift*. The decision boundary of the second data set $\mathcal{D}_2$ moves below that of the first data set, thus leading to an increase in the rate of *false positives*. The boundary of the third data set $\mathcal{D}_3$ moves above the initial one, thus leading to an increase in the number of *false negatives*.

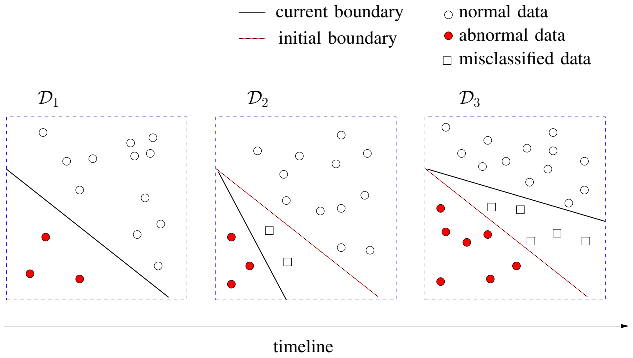

Fig. 2: Different types of concept drift relative to the initial data $\mathcal{D}_1$.

In practical terms, the problem of concept drift becomes highly dependent one what we define to be an anomaly. Suppose the current models are able to identify $\{a, b, c\}$ as being normal and $\{e\}$ as abnormal. A data point affected by *concept drift* would be similar to the existing one (e.g. $\{bb\}$) as opposed to one affected by *concept shift* (e.g. $\{d\}$) [6] which, depending on the application, could be considered a *true positive*. Hence, since no knowledge of $\{bb\}$ exists at the time of the training, this means we would require some sort of *incremental* updating of the models. A good approach will maintain, or improve the *accuracy* of the original models.

C. Modified OCSVM algorithm

Because anomaly detection is a one class problem, the notion of concept drift applies to the *normal* data. As such, for automating the incremental learning process, we devised an algorithm which analyses abnormal data, and tries to differentiate between *false positives* and *true positives* based on how *closely* the new instance matches the previous ones. In this way, we are updating the training data set (of normal instances) with the latest available information. The algorithm use the $OCSVM_{partial}$ decision function, majority voting, a window of OCSVM models and a heuristic function for selecting the best candidates to serve as training data for new models.

III. DeADA

The challenge of analysing big data in a timely fashion can be illustrated by having a thread *Receiver* receiving a series of network packets which are *enqueued* for analysis by a thread running the *OCSVM* algorithm. We set the size of the queue to be 6000 to reflect memory contraints. The OCSVM processing uses a model with 20000 support vectors with 20 attributes each. There are 80855 re-assembled packets (totalling 72 MB) in the data set which are being sent over to the *Receiver* at a rate of 2571.95 packets/s (corresponding to a throughput of 18Mbps). The *classification* rate of the *OCSM* thread is 461.36 packets/s. In real time, the cut off point (stop of processing) is quickly reached when the queue buffer is full. In order to cover the gap between the incoming rate and the analysis rate, packets need to be dropped not only diminishing the probability of detecting attacks, but also diminishing the OCSVM's accuracy when affected by concept drift.

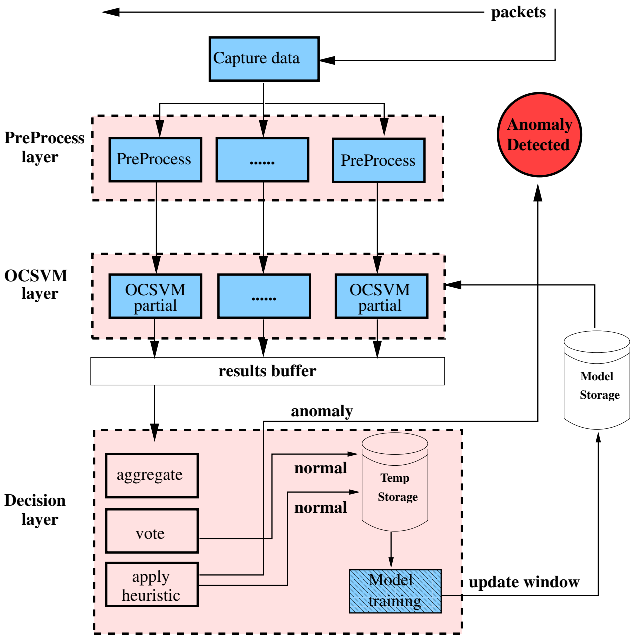

Fig. 3: DeADA architecture. Each blue box represents a computation node, at the various level. The *OCSVM layer* and *Decision layer* are the implementation of *USAE*. Note the *Model Training* node is stripped as this phase can be done via the architecture, or as an offline and separate process.

A. System Overview

We introduce DeADA, a dataflow architecture (Figure 3) designed as an end-to-end system (from capturing to analysis) structured on three layers: *PreProcess layer*, *OCSVM layer* and a *Decision layer*.

B. Hardware Architecture

The design challenge for a hardware OCSVM architecture is to efficiently transfer and process support vectors represented in sparse format. The algorithm details of OCSVM are given by Equation 2, and Figure 4 presents the customized FPGA architecture which uses our *modified OCSVM algorithm*. At each clock cycle, the ROM kernel and the RAM kernel feed the required data into the stream aligner, and the data-paths process the aligned data to generate one partial result per data-path.

The ROM kernel stores shared incoming instances and the RAM kernel distributes the support vectors loaded from off-chip memory. In an OCSVM system, some of these attributes may or may not exist for some instances, thus many implementations store the support vectors in sparse format as $[index, value]$ pairs, along with their corresponding

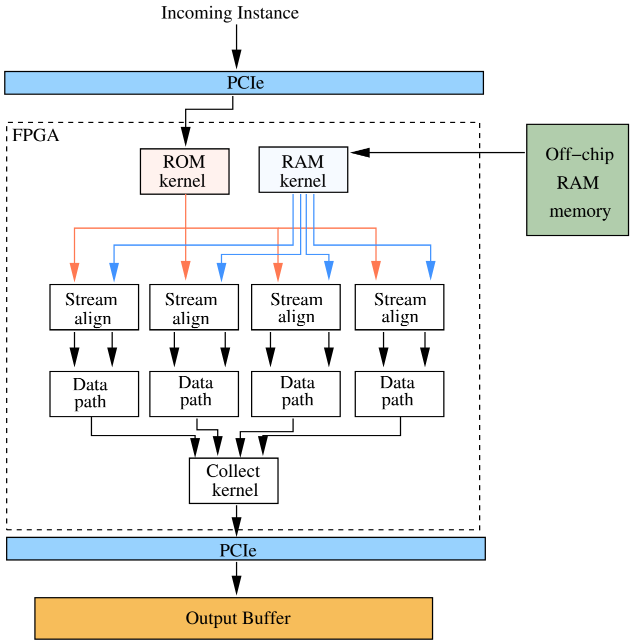

Fig. 4: FPGA architecture for OCSVM with USAE.

λ coefficients. Similarly the incoming instance is formed of $[index, value]$ pairs, where $index$ is the *ordinal* of the attribute. The support vectors and the shared instance are distributed across the *DeADA units*(a grouping of a stream aligner and a data-path). Support vectors from a model can be distributed across several units for quicker processing and several models can be processed on a unit (since the number of SVs from each model is known). Storing the shared instances in on-chip ROMs saves communication bandwidth as the instances do not need to be loaded repetitively from external devices. The RAM kernels request consecutive memory bursts to hide the off-chip memory latency and maximize the memory bandwidth.

$$K(x, x') = exp\left(-\frac{||x - x'||_2^2}{\gamma}\right) \tag{2}$$

The stream aligners pre-process the loaded data from the ROM kernel and the RAM kernel. Due to the sparsity of instance data and support vector data, the applied arithmetic operations depend on data indices. A typical OCSVM algorithm such as the one used by LibSVM has five run-time conditions with different operations on the x x' input streams, the final two conditions dealing with scenarios where only one of the streams has data left. If implemented as a data-path, all the possible arithmetic operations need to be implemented, to ensure correct functionality. Instead, we develop finite-state machines named stream aligners to pre-align the input streams. When stream indices are not even, a stream aligner fetches new data from the stream with the smaller index, and outputs 0 for the other stream. As an example, at cycle N of runtime, $x[i].index > x'[j].index$, the x' stream outputs its latest fetched data, and fetches $x'[j++]$ from the ROM kernel, while the x stream remains the same and outputs 0. With the stream aligners dynamically adjusting the stream output data, the five run-time condition are now implemented with the same arithmetic operations: $(x - x')^2$. In the previous example, when x is set to be 0 the arithmetic operation becomes x'^2. In an OCSVM kernel, the multiplication results are accumulated, generating a feed-back loop in the data-path. If floating-point arithmetic operators are used, multiple cycles are required to generate one result, and the feed-back loop will thus reduce the data-path throughput. We represent the input streams with fixed-point formats, to ensure the arithmetic operations can be finished within one clock cycle.

We define the overall data size ds as in Equation 3, where M are the total number of models, S_i the total number support vectors of the ith model, a_j the number of attributes of the support vector and $a_{instance}$ the number of attributes of the analysed instance.

$$ds = \sum_{i=1}^{M} \sum_{j=1}^{S_i} a_j + a_{instance} \tag{3}$$

IV. Evaluation and Interpretation

We measure the classification accuracy of the first data set and use that as the *reference accuracy* of the system. We assess the benefits of the *modified OCSVM algorithm* and DeADA, highlighting the importance of accelerated anomaly detection in maintaining the *reference accuracy*. For the experiment we have chosen the *Heartbleed* bug as the anomaly.

A. Setup

The compare the efficiency of the DeADA architecture with software implementations with single-device and multi-device systems. The CPU designs are parallelised under the OpenMP framework, and the data-paths in a DeADA architecture are replicated to fully exploit all available resources. We run the CPU designs on a Dell PowerEdge R610 machine, with 24 Intel(R) Xeon(R) X5660 cores running at 2.67GHz. For single-FPGA designs, we use a Max3 data-flow engine (with a Virtex-6 SX475T FPGA) from Maxeler Technologies, host in a MaxWorkstation. The multi-FPGA designs run on an MPC-C500 compute node with 4 MAX3 dataflow engines.

Heartbleed is a recent exploit of the OpenSSL implementation of the SSL protocol and takes advantage of a lack of bounds check in the payload of a Heartbeat request.

We use Wireshark [7], a tool for capturing and analyzing network traffic, to record ≈2000 packets of normal OpenSSL traffic.

To simulate drift we start from the initial 2000 packets, and continuously increasing the size of the `record.length` field until we have generated 400000 data points/packets. We randomly introduce the Heartbleed anomalies with a probability of 0.1.

In order to accelerate the OCSVM layer we have implemented the generic FPGA design from Figure 4 using the Maxeler platform. The core parts of the implementation are the **Stream aligner** and the **Data path**, which form a **DeADA unit**. We define the *replication factor* to be the number of DeADA units (e.g. Figure 4 has four DeADA units).

B. DeADA evaluation

In this section we will demonstrate the benefits of using DeADA and an accelerated *OCSVM* layer, in performing *live* network intrusion detection.

Table I contains the resource usages for the Maxeler implementation of the OCSVM layer. As we can see in both cases only ≈40% of the resources are being used. The design parallelism is limited by available memory bandwidth (38.4 GB/s for MAX3), and cannot be further increased to exploit the remaining resources. The memory bus width (3072/6144 bits) is a bottleneck when it comes to scaling.

Card	LUTs	FFs	BRAMs	DSPs	component
MAX3	21.97%	16.79%	0.94%	9.52%	data-paths
MAX3	43.25%	30.89%	30.31%	9.52%	total

TABLE I: Resource usage for a build with replication factor 48 on MAX3. Infrastructure includes PCIe and DRAM controllers.

As we have seen from section III a slow analysis/processing rate allows for a reduced number of packets to be analyzed in real time. Hence, we define $DropRatio$ (Equation 4) as a metric indicating the number of packets which need to be dropped before a *single* packet can be processed.

$$Drop\ Ratio = \frac{arrival\ rate}{processing\ rate} \tag{4}$$

Table II contains the resulting $DropRatios$ when accelerating the OCSVM layer of DeADA, for various replication factors. For replication factors 48 and 192 the design is tested on a MAX3 node. The formula for computing the $DropRatio$ is given by Equation 4. We define the theoretical execution time of a hardware design to be the performance when all N replicated data-path are working actively in runtime, generating N partial results each clock cycle. As shown in Table II, for the hardware designs, while the ROM kernel initialisation time dominates the overall execution time for small data sets, the measured execution time approximates the theoretical execution time when data size increases. Due to the higher processing capacity, the hardware DeADA designs achieve up to 6.17 times lower $DropRatio$ compared a 24 core software counterparts. Moreover, the DeADA designs are 4.7 to 10.7 times more efficiency than the software implementations. The power efficiency of the multi-FPGA design is limited by the power efficiency of its host CPUs, which consume 209 W in idle state.

	Drop Ratio				P	E (10^{-6})
CPU Cores	Data size (mil.)					
	0.4	3.2	25.6	102.4		
6	1.9	19.54	164.6	718.66	280	4.97
12	0.92	6.37	58.79	238.42	326	12.9
24	0.92	3.8	42.51	207.30	363	13.2
FGPA DeADA units						
48 (1-FPGA)	1.35	4.38	33.83	130.66	145	52.8
192 (4-FPGA)	0.33	1.12	8.64	33.5	482	61.9
theoretical (4-FPGA)	0.13	1.04	8.38	33.5	n/a	n/a

TABLE II: Drop ratios for a CPU server and a MPC-C500 system. P indicates power consumption in Watt, and power efficiency E is calculated as 1 / ($Dropratio \cdot P$).

In order to see how the accuracy decreases with time we split the 400000 data samples into 5 batches. For each batch, a varying number of packets are analysed depending on the $DropRatio$ (lower drop ratio means more packets are analysed). From Figure 5 we can see that a *high drop ratio* reduces DeADA's capability of handling the concept drift in the system. When subjected to concept drift $DropRatio$, the *simple* and *modified* (used by DeADA) algorithms lose information about the properties of the *normal data*, thus diminishing their discrimination abilities. This can happen in scenarios where the base models do not incorporate information about future data points, hence the data points behave as when subjected to *concept shift* (i.e. they act as true positives). However, if we lower the $DropRatio$ even further ($DropRatio = 33$) the accuracy is being maintained throughout the length of the experiment, as newer information is being incorporated into the models more frequently.

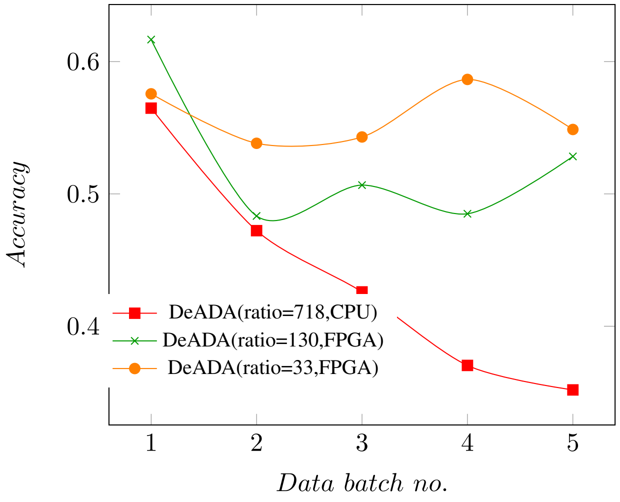

Fig. 5: Impact of various drop ratios on the accuracy of the classification.

V. Conclusion

This paper proposes DeADA, a solution for detecting anomalies in *live* network data. We evaluate the DeADA dataflow architecture built around the *modified OCSVM algorithm* on a case study of the Heartbleed bug in an environment with aggressive concept drift. Accelerating the anomaly detection not only allows for more data/packets to be analysed thus capturing more potential attacks, but it is also important in maintaining the accuracy of the system as can be seen from our evaluation.

Future work would focus on mapping both the *PreProcess* and *Decision* layers to FPGA to achieve a higher analysis rate, and evaluate DeADA on larger data sets captured from non-simulated environments.

Acknowledgement. This work is supported in part by UK EPSRC, by the European Union Seventh Framework Programme under Grant agreement number 257906, 287804 and 318521, by the HiPEAC NoE, by the Maxeler University Program, by Altera, and by Xilinx.

References

[1] B. D. W. Group, "Big data analytics for security intelligence," CLOUD SECURITY ALLIANCE, Tech. Rep., 2013.

[2] H. Yang and S. Fong, "Countering the concept-drift problem in big data using iovfdt," in *Big Data (BigData Congress), 2013 IEEE International Congress on*, June 2013, pp. 126–132.

[3] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt, "Support vector method for novelty detection." *NIPS*, vol. 12, pp. 582–588, 1999.

[4] D. M. Tax and R. P. Duin, "Support vector data description," *Machine learning*, vol. 54, no. 1, pp. 45–66, 2004.

[5] M. Papadonikolakis and C. Bouganis, "A scalable fpga architecture for non-linear svm training," in *ICECE Technology, 2008. FPT 2008. International Conference on*, Dec 2008, pp. 337–340.

[6] S. Wang, S. Schlobach, and M. Klein, "What is concept drift and how to measure it?" in *Knowledge Engineering and Management by the Masses*, ser. Lecture Notes in Computer Science, P. Cimiano and H. Pinto, Eds. Springer Berlin Heidelberg, 2010, vol. 6317, pp. 241–256. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-16438-5_17

[7] Wireshark. (2010, Jun.) Wireshark. http://www.wireshark.org/, accessed June 2014. [Online]. Available: http://www.wireshark.org/