Network Intrusion Detection Using Neural Networks on FPGA SoCs

Lenos Ioannou and Suhaib A. Fahmy
School of Engineering
University of Warwick, Coventry, UK
{l.ioannou, s.fahmy}@warwick.ac.uk

***Abstract*—Network security is increasing in importance as systems become more interconnected. Much research has been conducted on large appliances for network security, but these do not scale well to lightweight systems such as those used in the Internet of Things (IoT). Meanwhile, the low power processors used in IoT devices do not have the required performance for detailed packet analysis. We present an approach for network intrusion detection using neural networks, implemented on FPGA SoC devices that can achieve the required performance on embedded systems. The design is flexible, allowing model updates in order to adapt to emerging attacks.**

I. Introduction

The Internet of Things (IoT) is driving an exponential growth in connectivity between lightweight embedded systems. These devices are often severely computationally constrained, being designed to fulfil a single task well. This increased networking presents a challenge in terms of network security since these devices can expose a wider attack surface on account of not being as rigorously engineered as more complex systems. Indeed, the use of IoT devices as a tool in cyberattacks was exemplified by the Mirai malware in 2016, among other cases.

Traditional network security has aimed to provide confidentiality, integrity and availability of resources to authorized users. This has often occurred in more controlled network environments such as corporate networks, where firewalls serve as a secure point of interface with open networks. Even in such cases, the possibility of an internal system being compromised requires monitoring for attacks of all traffic, even from within the network.

Intrusion Detection Systems (IDSs) collect and analyse information from the systems within a network for malicious attack detection. Detection can be logged as an event of interest or trigger a defence mechanism to deal with the event in real-time. Mainstream IDSs use pattern matching, string matching, multi-match packet classification and regular expressions for operation [1]. These computationally complex approaches are often implemented using hardware accelerators on FPGAs or ASICs, or run on highly parallel multi-core processors or GPUs to enable them to process network traffic at the high rates required. Hence, such complex systems are usually integrated within the network infrastructure of large organisations.

The limited computing power of embedded systems means IoT devices will often not incorporate significant security capabilities at the nodes, making them an ideal target for malicious attacks. With such devices being deployed in less controlled environments, and without access to significant infrastructure, more lightweight approaches to such security mechanisms are required.

In this paper we explore a Network Intrusion Detection approach based on Machine Learning (ML), specifically Neural Networks (NNs), that provides flexibility to evolve to emerging attacks. We demonstrate how this can be implemented on a lightweight Xilinx Zynq FPGA SoC, designed to act as an IoT gateway, that processes packets at line rate while enabling model parameter updates to adapt to changing requirements.

II. Background

Intrusion Detection Systems (IDSs) can be divided into two categories, according to the detection method used:

- **Signature (or misuse) based:** Captured data is compared against a database containing signatures of known attacks.
- **Anomaly based:** Captured data is compared against a model of the expected normal behaviour of the system. If a deviation is observed then an attack has been detected.

Signature based IDSs are widely used in commercial systems because of their accurate detection of known attacks, while anomaly based systems are prone to generating false classifications. Signature based IDSs, however, fail to detect unknown (zero-day) attacks. There can also be a significant delay for a new attack to be detected and its signature generated and distributed in an update [2]. Moreover, signature based systems must consider a large database of signatures, requiring substantial memory and computational power. Hybrid implementations of signature and anomaly based IDSs present a more robust approach since one method complements the other, though these still require significant computing power.

Intrusion Detection has been an appealing domain for machine learning algorithms in general. The strongest incentive lies in the ability of ML algorithms to generalize their learned pattern to new, unknown data. As a result, algorithms in this domain have the potential to detect modified known attacks, that have been altered adequately to deceive signature based systems, in addition to unknown, zero-day, attacks. It is also worth considering that IoT, as a developing domain, will entail evolving (normal) traffic patterns as it finds more uses, so the safe patterns of communication are themselves evolving, and hence an adaptable approach to intrusion detection is needed.

As the functionality of machine learning models is defined during training, the dataset used becomes very important. Algorithms in this domain extract patterns from the training dataset that are subsequently used to classify new, previously unseen data. Datasets used for intrusion detection fall into two broad categories, private (or custom) and public datasets. Private datasets may contain more realistic data for training and testing as in most of the cases they are created from the specific scenario under consideration. Moreover, they can be tailored to a specific attack detection by manipulating the number of records in each class accordingly. Proprietary and commercially sensitive datasets, however, are not available to researchers. Publicly available datasets, on the other hand, are widely used and, as a result, thoroughly tested [3]. They constitute a safer choice to avoid potential flaws and, more importantly, they provide a means to compare with previous work using the same datasets.

III. Related Work

Network security has sustained interest in the research community and IDSs using a variety of approaches have been proposed. Acceleration of pattern matching on FPGAs has been explored in [4, 5]. The authors in [6] proposed an approach using Principal Component Analysis (PCA) with features extracted from network traffic, which was tested on the publicly available KDD Cup 1999 dataset. The IDS was implemented on a Xilinx Virtex II Pro FPGA and achieved a 23.76 Gb/s throughput with an attack detection rate of over 99%. In [7], the authors explored an energy efficient implementation on an Altera Cyclone IV using Decision Trees (DTs). The authors present two test cases: the first classifies the NSL-KDD dataset using 9 manually selected features out of 41, achieving a 96.5% accuracy on the train set and 77.8% on the test set. The second detects probe attacks on a custom dataset, misclassifying only 3 out of 37548 instances in the test set. The hardware implementation of the probe attack detection DT is 15.4× better in throughput while consuming only 0.03% the energy of its software equivalent executed on an Intel Atom CPU. The authors further expanded their work using their custom dataset to evaluate 3 ML classifiers (Decision Tree, Naive Bayes, and k-Nearest Neighbours) in a similar manner [8]. In this case, the fastest classifier in hardware was 926× faster while consuming 0.05% the energy of its equivalent in software. The work in [9] showed how security primitives could be built into network controllers to enable enhanced security.

In broader work in neural network implementations, the NN developed in [10] detects Distributed Denial of Service (DDoS) and DoS attacks. The authors use a custom dataset to train a three-layer (shallow) NN for binary classification (normal-DoS/DDoS) and test it in a simulated IoT network, demonstrating 99.4% accuracy. The work in [11] presents two NNs trained on the UNSW-NB15 and NSL-KDD datasets to detect DoS attacks. The authors use only input features relevant to such attacks, obtaining at best a detection accuracy of 99% on the NSL-KDD and 97% on the UNSW-NB15.

The work in [12] presents two NNs trained on the NSL-KDD dataset to detect all 4 types of attacks in the dataset (DoS, Probe, R2L and U2R). The first NN categorizes records between normal and malicious (binary classification), while the second classifies the malicious records into types (5-categories). On the test set, for binary classification, the best accuracy of 81.2% was obtained using a subset of the input features, while for attack classification the best accuracy of 79.9% was obtained using all features. The work in [2], similarly uses two shallow NNs trained on the KDD Cup 1999 dataset for binary and attack type classification. The NNs use 36 of the 41 features, demonstrating an average precision of 98.86% for binary classification and 95.05% for the attack type classification.

The authors in [13] review IDSs that utilize deep learning approaches, the most relevant of them to our work found in [14, 15, 16]. The work in [14] uses a Recurrent Neural Network (RNN) to the NSL-KDD dataset using all provided input features for binary and attack type classification. For binary classification, the authors obtained 99.81% and 83.28% accuracy on the train and test set respectively using 80 hidden nodes. Regarding the 5-category classification, they demonstrate 99.53% and 81.29% accuracy on the train and test set respectively using 80 hidden nodes. Although Convolutional Neural Networks (CNNs) are primarily used for image classification tasks, an approach that uses CNNs to classify the NSL-KDD dataset is proposed in [15]. The authors utilize an image conversion technique that maps all the input features of each record to an image. They implement 2 popular CNN models, ResNet 50 and GoogleNet, obtaining 79.14% and 77.04% on the test set for binary classification respectively. The Deep Neural Network (DNN) approach presented in [16] uses 6 raw features of the NSL-KDD dataset to achieve 91.62% and 75.5% accuracy on the train and test set respectively. Using the same number of raw features, the authors applied their methodology to deep RNNs in [17] obtaining 89% accuracy on the test set.

The topology configurations of the aforementioned NNs are summarised in Table I, where available.

TABLE I: Network configurations in related work.

Citation	Configuration
[16]	6-12-6-3-2
[10]	6-3-1
[11]: UNSW	6-7-1
[11]: NSL-KDD	5-6-1
[12]	29-21-2
[12]	41-23-5

IV. Experimental Methodology

A. Neural Networks

Neural networks are computational models inspired by human cognition, able to model complex non-linear functions that correlate inputs and expected outputs. They have successfully been applied in a broad range of fields, from automotive [18], to healthcare [19]. Each neuron calculates the weighted

sum of its inputs and adds this to an offset value (bias), passing the result to an activation function. NNs comprise a sequence of neuron layers propagating results between them. During *training*, the functionality of the network is defined using back-propagation to determine the weights and biases that optimally classify the training data. Once these are set, the NN can be used for *inference* where it processes new data to determine a classification. Inputs to the NN are represented numerically and so symbolic (or categorical) inputs must be converted to suitable formats before being applied. An indicative structure of a NN is shown in Fig. 1.

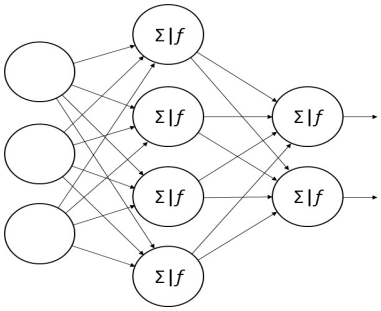

Fig. 1: Neural network structure.

Training can be very time consuming and typically occurs offline using highly parallel computing systems. Trained model parameters are used to perform online classification, and as such, inference hardware needs to be optimised. We use a three-layer (shallow) NN for intrusion detection in this work. While DNNs and RNNs are growing in importance, they entail more complex computation making the real time processing on embedded hardware extremely difficult.

In Section III we saw that tailoring a NN to detect only a single type of attack or all the attacks in one category can result in better accuracy. Moreover, selecting the most relevant features from the dataset decreases the dimensionality and this in turn enables NNs to perform better in simple classification [12]. Hence, the proposed NN is trained to detect all types of attacks in one category (Normal-Anomaly), using a selected subset of the available features.

B. NSL-KDD Dataset

We use the publicly available NSL-KDD dataset, a labelled dataset for supervised learning. It is an updated version of the KDD Cup 1999 dataset, addressing its shortcomings [20]. While the dataset is not directly related to IoT applications, it is widely used, enabling comparisons with previous work. The approach presented here can be applied to any future dataset which can be used to retrain the network for IoT specific traffic patterns. The dataset is divided into the *train* and *test* sets which contain data for normal and malicious traffic. Each entry comprises 41 features categorized into 3 groups [20]:

- **Basic features:** features that are extracted from a TCP/IP connection.
- **Traffic features:** features that are generated within a window of the last 100 connections, to enable detection of longer probe attacks. These features provide an element of time-domain memory. Traffic features are further categorized into service and host based.
- **Content features:** features that are extracted from the packets' data and provide the means to detect attacks with infrequent sequential patterns.

The train set contains 22 attack types, divided into 4 main categories: DoS (Denial of Service), Probe, R2L (Remote to Local) and U2R (User to Root). In the test set, there are 17 additional attacks that fall into the same 4 categories. In this way, the ability of the NN to generalize its learned pattern to unknown data is put under test.

In order to fairly train the model, categorical features are mapped to one-hot encoded representation for the training phase, mitigating the possible bias introduced by ad-hoc numerical mapping.

C. Software Implementation

We used TensorFlow [21] to train a NN with 29 input features, 21 hidden neurons and 2 output neurons, similar to that in [12]. Of the 41 input features, the authors in [22] concluded that 8 of them have little or no impact in attack detection, while the work in [12] highlights that the values of 4 other features are close to 0. The selected features span all types of features in the dataset. This enables the NN to extract patterns in the time domain using Traffic Features, thus avoiding the use of more computationally complex machine learning models that do so with raw features, such as RNNs. Out of the 29 selected features, 3 are categorical, increasing the number of inputs to 110 after one-hot encoding. We use the relatively simple and inexpensive Rectified Linear Unit (ReLU) activation function, that can be easily implemented with a comparator instead of more complex functions that include divisions and exponents, such as the sigmoid function.

The proposed NN was trained with the Adam optimizer, using the cross entropy loss function (that also includes softmax) with weights and biases randomly initialized. We further determine some training parameters experimentally, such as the learning rate and batch size. For fair comparison, the same randomly initialized weights and biases are used in all our experiments. We ran our experiments using 3 learning rates (0.01, 0.001, 0.0001) on four different batch sizes (32, 64, 128, 256) for a total of 5 epochs. We evaluate the classification performance of each run using:

$$accuracy = 100 * \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

where:

- **TP:** True Positive, corresponds to an attack that has been correctly detected.
- **TN:** True Negative, corresponds to normal traffic that has been correctly classified as such.
- **FP:** False Positive, corresponds to normal traffic that has been classified as an attack.
- **FN:** False Negative, corresponds to an attack classified as normal traffic.

We summarize the highest accuracies obtained after the pass of one epoch in Table II. While we train the proposed NN on the train set and test it on the test set, the results in Table II are selected by prioritizing the accuracy obtained from the test set across runs.

TABLE II: Accuracy results for training parameters.

Batch Size	Learning Rate					
	0.01		0.001		0.0001	
	Test	Train	Test	Train	Test	Train
32	77.61	89.15	**80.52**	**96.02**	80.37	89.09
64	73.16	94.71	80.64	94.05	80.29	93.62
128	76.65	93.09	79.01	96.62	79.80	91.99
256	77.56	94.49	80.84	94.22	77.47	94.06

From the results in Table II, we see that learning rate of 0.001 and batch size of 32 provides the optimal combined accuracy across the test and train sets. This results in the confusion matrix of the test set in Table III.

TABLE III: Test set classification results.

Predicted Class	Actual Class	
	Normal	Malicious
Normal	9257	3937
Malicious	454	8896

Compared to other works in the literature that use the NSL-KDD dataset, with which a direct comparison can be made, the proposed model is close to that in [12], where the authors reported 99.3% and 81.2% accuracy on the train and test set respectively. It is worth noting that the authors in this case normalized the dataset prior to its use. While data normalization has been proven to enhance the accuracy of NNs, it also entails additional workload during inference. The authors in [16] use a DNN with 6 input features, reporting 91.62% and 75.75% accuracy on the train and test set respectively. This shows that deep models that use a small subset of the input features do not necessarily outperform shallow models that use more features. All the referenced implementations in this paper that use the NSL-KDD dataset are summarized in Table IV, along with their configurations.

TABLE IV: Accuracy comparisons on the NSL-KDD dataset.

Citation	ML Model	Classification	# Features (out of 41)	Accuracy %	
				Train Set	Test Set
[7]	DT	N/A	9	96.5	77.8
[12]	NN	Binary	29	99.3	81.2
[12]	NN	5-Cat.	41	98.9	79.9
[16]	DNN	Binary	6	91.62	75.75
[17]	D-RNN	Binary	6	N/A	89
[14]	RNN	Binary	41	99.81	83.28
[14]	RNN	5-Cat.	41	99.53	81.29
[15]	CNN-ResNet50	Binary	41	N/A	79.14
[15]	CNN-GoogleNet	Binary	41	N/A	77.04
Proposed	NN	Binary	29	96.02	80.52

D. Hardware Implementation

Unlike previous work, our aim is to build a fully functional embedded IDS to perform these classifications in real time. Hence, the trained NN was used to build a working system for this purpose. We used Xilinx Vivado HLS 2016.4 targeting the Xilinx Zynq Z-7020 FPGA as found on the Xilinx Zedboard. This is a modest FPGA SoC device that includes flexible Programmable Logic (PL) tightly coupled with an Arm Cortex-A9 Processing System (PS), as shown in Fig. 2. This system is designed to act as an IoT gateway, securing the network for a group of less capable devices. The peripherals, e.g. Ethernet Phy and SD card, are connected through Multiplexed I/O (MIO) interconnect to the Arm core and 512 MB of DRAM is also available. This flexible connectivity enables runtime processing of network data by forwarding packets to the accelerator, or processing them in software. For testing and verification, it allows the test set and model coefficients to be stored in an SD card, transferred to the memory and then to the accelerator over DMA.

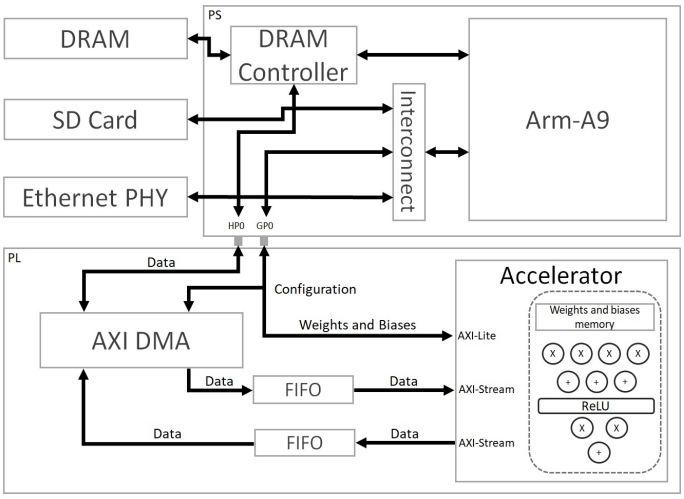

Fig. 2: Overview of the Xilinx Zynq based system architecture.

Most work on optimising FPGA implementations of neural networks considers fixed network parameters. Network pruning, data quantization, and reduced arithmetic precision can be exploited to trade off performance, power consumption, and detection accuracy [23, 24, 25]. However, this comes at the cost of flexibility as any change in network parameters requires a new design exploration and hardware implementation process. The architecture we present is designed to be flexible, by allowing the coefficients to be modified at runtime, thereby enabling the same hardware to be used to detect different or evolving attacks without the need for additional design space exploration or hardware optimisation.

Vivado HLS allows us to exploit the inherent parallelism in the NN structure using pragmas to unroll loops for maximum parallelism and performance, without the need for low level Hardware Description Language (HDL). The inputs and intermediate results are represented in single precision floating point format (IEEE-754), as the architecture is designed to retain flexibility to accept newly trained model parameters. The accelerator operates in one of three modes: IDLE, LOAD, or COMP. It starts in the IDLE mode where it can make a transition to LOAD or COMP. Transitions between states are

triggered from the Arm core over AXI-Lite since these are not time critical operations.

In LOAD mode, the coefficients (weights and biases) of the model are modified to update the NN at runtime, which is done over AXI-Lite using the 4 accelerator inputs:

- **mem_sel:** selects the memory bank to configure. (i.e. first layer weights, first layer biases, etc.)
- **dimA:** indexes the first dimension of the weights, or the only dimension of the biases.
- **dimB:** indexes the second dimension of the weights.
- **coeff_in:** value of the coefficient to be stored.

FPGAs support flexibility through reconfiguration by loading alternative bitstreams that modify the hardware on the FPGA [26]. One method for using different NN models would be to generate multiple bitstreams and load them as needed. However, this would entail the separate design and compilation of these optimised hardware models and would not allow for easy modification of model parameters to deal with emerging attacks. The Xilinx Zynq allows the PL configuration to be changed by the PS in software, taking around 30 milliseconds. Alternatively, Partial Reconfiguration (PR) can be used with an optimised reconfiguration controller [27] to reduce the time to below 10 milliseconds. We choose to retain full flexibility by implementing a general datapath with reprogrammable coefficients, rather than tightly optimising the datapath around a fixed set of coefficients.

We measured the time needed for the configuration of all 2375 coefficients to be 2.273 ms. This includes the time needed for the Arm core to iterate through the data, increment its indexing variables, configure the accelerator accordingly and make the appropriate checks. Updating coefficients is not considered a time-critical operation as one configuration of the IDS is expected to be active for a large volume of network data. Nonetheless, the proposed approach offers competitive configuration time compared to reconfiguring the hardware, while offering a much more flexible implementation that allows coefficients to be updated directly, without the need for vendor tools and a full hardware compilation.

The intrusion detection process takes place in the COMP state. To mitigate the increased complexity due to the one-hot encoding, we take advantage of the fact that only one input of each one-hot encoded feature is used at a time. During inference, integer representation is used for each attribute and in each case only the index of the active attribute is needed. The index of the active attribute is used as an address to a Look-Up-Table, that outputs the corresponding weight. This restores the number of input features needed for inference from 110 to 29, while also avoiding redundant multiplications by 0 caused by the inactive attributes in each one-hot encoded feature. Meanwhile, any multiplication by 1 of each active attribute is replaced with a low latency table look-up. The 29 input features along with the 2 output results (corresponding to the normal/malicious score), are interfaced with the Arm core through 2 separate AXI-Stream ports with the data transferred sequentially in consecutive clock cycles.

The timing results of the implemented design from HLS are shown in Table V while the resource utilization on the Xilinx Zynq device is shown in Table VI.

TABLE V: Timing results on the Xilinx Zynq Z-7020.

Frequency (MHz)	Latency (Clock Cycles)	Initiation Interval (Clock Cycles)
76	237	29

The initiation interval of 29 clock cycles is bounded by the number of input features that need to be read through the AXI-Stream port.

TABLE VI: Resource utilization on the Xilinx Zynq Z-7020.

	LUTs	FFs	DSPs	BRAM
Utilized	26463	56478	111	88
Available	53200	106400	220	280
% Utilization	50	53	50	31

The proposed system, shown in Fig. 2, uses 2 FIFOs on each AXI-Stream port of the accelerator to act as buffers. Data is transferred to and from the AXI-Stream ports through the AXI-DMA that is interfaced with the PS using the HP0 (High Performance 0) port. The HP0 port, in turn, using the DRAM controller, transfers data to and from DRAM. The configuration of the accelerator coefficients as well as the configuration of the AXI-DMA take place using AXI-Lite ports, which are interfaced with the PS through the GP0 (General Purpose 0) port.

V. Results and Evaluation

To evaluate the performance of the proposed IDS in practice, we used Xilinx Vivado 2016.4 to implement the system as shown in Fig. 2. The AXI-Timer IP, operating at 100 MHz, was used to measure the execution time. The coefficients and test dataset were stored in an SD card, read from the Arm core, transferred to DRAM and then fed to the accelerator first by configuring the coefficients in LOAD mode, before reading the dataset in COMP mode. In order to provide a reference for comparison, the execution time of the proposed IDS on a single core of the Arm-A9 (bare metal) was recorded. To demonstrate and evaluate the benefits of utilizing the Look-Up-Table mechanism, we also provide the execution time of the unoptimized software implementation on the Arm core. This version of the NN uses 110 inputs at the input layer and goes through a number of redundant multiplications as described in the Section IV-D. We also compare our work with the test time of the NN used in [11]. Although the authors use a different dataset and detect only DoS attacks, we only focus on the execution time and corresponding workload in this section. Their implementation uses Keras and Theano frameworks in Python, on an Intel Core i3 2.4GHz under Debian Linux 8. The authors used 43748 records to test their NN. For a fair comparison, we normalize their obtained test time of 0.466

seconds according to the number of test records in the NSL-KDD (22544). The execution time of the four methods for the classification of the test set is shown in Table VII.

TABLE VII: Execution time.

Arm-A9 [a] @667MHz	Arm-A9 [b] @667MHz	Accelerator [b] @76MHz	Idhammad et al. [11] (normalized)
4751.440ms	1458.1ms	9.018ms	240.136ms

[a] Unoptimised, 110 inputs.
[b] Optimised, Look-Up-Table.

The execution time of the accelerator includes the time needed for the input data to be transferred to the accelerator from the DRAM and the results to be written back to the same memory. The utilization of a Look-Up-Table mechanism yields a 69% reduction in the software execution time. A straightforward comparison between the proposed accelerator, operating as a streaming engine in this case, and the optimised execution on the Arm Cortex-A9 shows a 161.7× improvement in the execution time. Compared to the unoptimized software version, the HW implementation is 526.9× faster.

If we naively compare the execution time of the proposed NN, considering our proposed optimised implementation with a 29-21-2 configuration, and the work in [11], our proposed accelerator performs about 26.6× faster. Meanwhile, our optimised model on the Arm core is about 6× slower. In this case, however, the workload of the NN in [11] with a 6-7-1 configuration is significantly smaller compared to the proposed NN's configuration. Taking into account the number of multiplications and additions in each layer, as those are the most computationally intensive operations, we estimate 49 multiplications and 57 additions for the NN in [11]. Meanwhile for our work, a total of 651 multiplications and 674 additions are required. This amounts to 13.3× the multiplications and 11.8× the additions of the NN used in [11], while delivering 26.6× its performance. Overall, our proposed approach is able to detect more types of attack: DoS, Probe, R2L and U2R, at a faster detection rate compared to the work in [11].

A. Network Throughput and Detection Rate

A considerable aspect of an IDS is whether it can make decisions on packets at a suitable rate to ensure detection does not lag the start of an attack significantly. Ideally, such a system should be able to flag malicious packets before many of them have entered the network, so that evasive action can be taken. We determined the time required to classify a single data record on both the Arm core and the accelerator by normalizing execution time in Table VII. We take into consideration the required minimum transmission size for IPv4 which is 576 bytes according to the Internet Protocol [28]. We observe the results in Table VIII.

At 1Gbps, 217,014 packets per second of the minimum packet size can be transferred when the network is saturated.

TABLE VIII: Detection rate in packets.

Transfer Rate (Packets/Second)	Platform	Latency (µs)	Detection Rate (Packets/Classification)
1Gbps	Arm-A9	64.678	14.036
(217,014)	Accel	0.4	0.0868
10Gbps	Arm-A9	64.678	140.360
(2,170,139)	Accel	0.4	0.8680

The accelerator offers a detection rate within a small fraction of a packet (0.0868 packets). On the other hand, the Arm core can only process one in 14 packets. While the Zedboard does not offer a 10G Ethernet interface, we also evaluated the performance for such a setup that might be deployed in an edge datacenter interacting with IoT devices. Newer Zynq UltraScale+ development boards do offer 10G Ethernet, meaning our design could be ported to such boards for more complex networks. At 10Gbps, a maximum of 2,170,139 packets per second can be transferred. The detection rate in this case is still within a single packet (0.8680 packets), which is 16.2× faster than the Arm core at 1Gbps and 161.7× faster at 10Gbps. The Arm core at 10Gbps only processes one in 140 packets.

These results demonstrate the benefit of our hardware accelerated NN detection mechanism in terms of scaling to faster networks, while still offering the flexibility needed to accept updated model parameters for emerging threats. Porting to newer FPGA SoC devices such as the Zynq UltraScale+ would also likely offer significant runtime improvements.

VI. Conclusion

This paper presents an approach for network intrusion detection using NNs on FPGA SoCs. The topology of the NN maintains moderate computational complexity for a hardware implementation that can be deployed on a modest Xilinx Zynq device. It also allows runtime configuration of neural network parameters to allow for updates to address emerging attacks We used TensorFlow [21] to train the proposed NN using the NSL-KDD dataset, obtaining at best 80.52% accuracy on the test dataset. The proposed hardware accelerator is 161.7× faster than software execution on the Zynq Arm core, allowing it to detect malicious packets within a single packet window for 1Gbps and 10Gbps. In the future, we plan to investigate deeper and alternative neural network topologies for comparison in terms of performance and accuracy, as well as extending network testing to alternative datasets and more varied live traffic patterns in 10G networks. Finally, we aim to explore approaches to reduce latency by processing data directly in the PL to circumvent the PS network stack using the method in [29].

Acknowledgement

This work was supported in part by an IBM Faculty Award and the UK Engineering and Physical Sciences Research Council (EPSRC), grant EP/N509796/1.

REFERENCES

[1] R. Abdulhammed, M. Faezipour, and K. M. Elleithy, "Network intrusion detection using hardware techniques: A review," in *Proc. IEEE Long Island Systems, Applications and Technology Conference*, 2016.

[2] B. Subba, S. Biswas, and S. Karmakar, "A neural network based system for intrusion detection and attack classification," in *Proc. National Conference on Communication*, 2016.

[3] A. Gharib, I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "An evaluation framework for intrusion detection dataset," in *Proc. International Conference on Information Science and Security*, 2016.

[4] S. Yusuf, W. Luk, M. K. N. Szeto, and W. Osborne, "Unite: Uniform hardware-based network intrusion detection engine," in *Reconfigurable Computing: Architectures and Applications*, 2006, pp. 389–400.

[5] R. Proudfoot, K. Kent, E. Aubanel, and N. Chen, "Flexible software-hardware network intrusion detection system," in *Proc. International Symposium on Rapid System Prototyping*, 2008, pp. 182–188.

[6] A. Das, D. Nguyen, J. Zambreno, G. Memik, and A. Choudhary, "An FPGA-based network intrusion detection architecture," *IEEE Transactions on Information Forensics and Security*, vol. 3, no. 1, pp. 118–132, 2008.

[7] A. L. P. de Franca, R. P. Jasinski, V. A. Pedroni, and A. O. Santin, "Moving network protection from software to hardware: An energy efficiency analysis," in *Proc. IEEE Computer Society Symposium on VLSI*, July 2014, pp. 456–461.

[8] A. L. P. de Franca, R. P. Jasinski, P. Cemin, V. A. Pedroni, and A. O. Santin, "The energy cost of network security: A hardware vs. software comparison," in *Proc. International Symposium on Circuits and Systems (ISCAS)*, 2015, pp. 81–84.

[9] S. Shreejith and S. A. Fahmy, "Security aware network controllers for next generation automotive embedded systems," in *Proc. Design Automation Conference (DAC)*, 2015.

[10] E. Hodo, X. Bellekens, A. Hamilton, P. L. Dubouilh, E. Iorkyase, C. Tachtatzis, and R. Atkinson, "Threat analysis of iot networks using artificial neural network intrusion detection system," in *Proc. International Symposium on Networks, Computers and Communications*, 2016.

[11] M. Idhammad, K. Afdel, and M. Belouch, "DoS detection method based on artificial neural networks," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 4, 2017.

[12] B. Ingre and A. Yadav, "Performance analysis of NSL-KDD dataset using ANN," in *Proc. International Conference on Signal Processing and Communication Engineering Systems*, 2015, pp. 92–96.

[13] K. Kim, M. E. Aminanto, and H. C. Tanuwidjaja, *Network Intrusion Detection using Deep Learning: A Feature Learning Approach*. Springer Singapore, 2018, pp. 35–45.

[14] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21 954–21 961, 2017.

[15] Z. Li, Z. Qin, K. Huang, X. Yang, and S. Ye, "Intrusion detection using convolutional neural networks for representation learning," in *Proc. International Conference Neural Information Processing*, 2017, pp. 858–866.

[16] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *Proc. International Conference on Wireless Networks and Mobile Communications*, 2016, pp. 258–263.

[17] T. Tang, S. Zaidi, D. McLernon, L. Mhamdi, and M. Ghogho, "Deep recurrent neural network for intrusion detection in SDN-based networks," in *Proc. International Conference on Network Softwarization*, 2018.

[18] S. Shreejith, B. Anshuman, and S. A. Fahmy, "Accelerated artificial neural networks on FPGA for fault detection in automotive systems," in *Proc. Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 37–42.

[19] O. Karan, C. Bayraktar, H. Gümükaya, and B. Karlk, "Diagnosing diabetes using neural networks on small mobile devices," *Expert Systems with Applications*, vol. 39, no. 1, pp. 54–60, 2012.

[20] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE International Conference on Computational Intelligence for Security and Defense Applications*, 2009, pp. 53–58.

[21] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/

[22] K. Bajaj and A. Arora, "Improving the intrusion detection using discriminative machine learning approach and improve the time complexity by data mining feature selection methods," *International Journal of Computer Applications*, vol. 76, no. 1, pp. 5–11, August 2013.

[23] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions," *ACM Computing Surveys*, vol. 51, no. 3, pp. 56:1–56:39, 2018.

[24] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec 2017.

[25] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[DL] A survey of FPGA-based neural network inference accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 1, pp. 2:1–2:26, Mar. 2019.

[26] K. Vipin and S. A. Fahmy, "FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications," *ACM Computing Surveys*, vol. 51, no. 4, pp. 72:1–72:39, Jul. 2018.

[27] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient partial reconfiguration management on the Xilinx Zynq," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 41–44, Sep. 2014.

[28] J. Postel, "Internet protocol," *RFC*, vol. 791, pp. 1–51, 1981.

[29] S. Shreejith, R. A. Cooke, and S. A. Fahmy, "A smart network interface approach for distributed applications on Xilinx Zynq SoCs," in *Proc. Field Programmable Logic and Applications (FPL)*, 2018, pp. 186–190.