AMD
VERSAL

White Paper

USE OF AI AND MACHINE LEARNING CLASSIFIERS FOR DISTRIBUTED DENIAL OF SERVICE IN NETWORK SECURITY

INTRODUCTION

This white paper discusses the methods to prevent DDoS attacks in network switches, routers, and enterprise firewalls using a classification-based machine learning algorithm. Traditional methods of DDoS mitigation which include access control lists (ACLs) for packet filtering based on IP address, MAC address and other packet parameters or application monitoring using software algorithms to prevent zero-day attacks may be too slow in detecting the DDoS attacks and may also consume significant compute resources. In this paper we describe the implementation of random forest AI algorithm used as inference for traffic classification and predicting the malicious traffic which can cause a DDoS attack. The random forest algorithm is trained offline using some of the publicly available datasets described in the subsequent sections. The algorithm is implemented in programmable logic on an AMD Versal™ "adaptive SoC-based reference platform. A trained random forest model is inputted with DDoS traffic features over the PCIe interface, and the inference results are read by the CPU from DRAM memory attached to the AMD Versal device.

DDoS AND DoS: TYPE OF ATTACKS

When compute resources on a server in a data center or enterprise network are exhausted and it cannot service the authenticated access of a legitimate user, the scenario is defined as denial of service (DoS) when more than one host is involved in an attack it is termed as distributed denial of service (DDoS). A DDoS attack on a network can be performed by manipulating the parameters of the data link layer (Layer-2), network layer (Layer-3), transport layer (Layer-4) or higher layers (Layer 5-7). A DDoS attack on a server can be made by opening many connections to the target server, sending a large number of packets or with specific traffic patterns. Some of the traffic parameters which can cause the DDoS attack are described in the table below.

OSI LAYER	TYPE OF TRAFFIC	DDoS TYPE
Layer-2 (data link layer)	Ethernet frames	MAC Flood – broadcasting the Ethernet frames through a network switch, causing the switch traffic to go up exponentially and increase in MAC address learning requirement.
Layer-3 (Network Layer)	IP Packets	Flooding through ICMP packets – ICMP packets are ping packets targeted to an IP address
Layer-4 (Transport Layer)	TCP and UDP segments	Syn Flood – opens many TCP connections to target server causing exhaustion of all the server resources
Layer 5-7	Application protocols (https, ftp, http, SMTP etc.)	File downloads, image or video uploads and downloads, badly formed SSL requests with too many encryption/ decryption requirements

TRADITIONAL METHODS TO PREVENT THE DDoS

Since DDoS attacks can go from the physical layer to the application layer, the scope of the attack profiles and corresponding prevention mechanisms is wide. Below are some of the classical prevention mechanisms adopted to prevent DDoS attacks. An unresponsive network node (switch, router, or server) because of physical layer faults such as hardware or link failure, can also be inferred as DDoS issue and resolution may involve fixing the faulty hardware or fiber. The higher layer DDoS attack-prevention mechanisms can be more complex and may involve sophisticated traffic analysis.

1. MAC (data link) layer can limit the size of learning tables so it can only accept limited number of address requests. Also, the MAC address to be learned on any ports must be authenticated using authentication server and filtered accordingly.
2. Ping (ICMP) traffic corresponding to IP (network) layer must be rate limited using the software or hardware-based mechanism to avoid the ping flooding.
3. TCP or UDP based DDoS attacks such as SYN flooding or smurf attacks are prevented using blackholing, in which all the legitimate and malicious traffic is routed in a blackhole and dropped. UDP is connectionless so peer does not need to be notified while for a TCP protocol the peer needs to be notified when blackhole filtering is used.
4. At the session and presentation layers, the encrypted traffic for the TLS connections is decrypted and inspected against the traffic policies as attackers use SSL to tunnel HTTP requests to the server. The decryption may be resource consuming so offloading on hardware may be useful.
5. For application layer-based DDoS attack mitigation, application monitoring using a set of search algorithms is applied. The most common methods deployed are flow analysis, flow telemetry using the ACLs, and Web application firewalls.

None of the above methods use the machine learning algorithms for mitigation of DDoS attacks, Also they are more preventive instead of predictive. Many of the next-generation firewalls deploy some machine learning techniques using software-based inference models for network intrusion prediction and prevention. In the sections below we discuss one of the classifier algorithms (random forest) implemented in programmable attached network hardware for prediction of DDoS attacks.

AI MODELS FOR NETWORKING AND MODEL SELECTION

Networking has not escaped the AI trend and AI is being used both for optimization and decision making. AI learning models can be classified as supervised, unsupervised, semi-supervised and reinforcement. Choice of a particular model type is based on the problem statement, resources in terms of data input, and end goal in terms of performance metrics. The datasets mentioned in subsequent sections are all labeled data and make a perfect choice to use a supervised learning classifier for network security use cases. A supervised learning classifier maps input variables x to the output class label y based on the labeled training data. The mapping function is then used to predict the classes that a new observation belongs to. Support Vector Machine (SVM), Decision Tree (DT), Artificial Neural Network (ANN), Naïve Bayes classifier (NB), Logistic regression, K Nearest Neighbors, and random forest are examples of a few supervised machine learning classifiers. An ensemble-based random forest classifier was chosen for DDoS implementation and has produced impressive performance metrics and ease-of-scalability HW implementation.

RANDOM TREES – MODEL OVERVIEW AND USAGES

Random forests or random decision forests are an ensemble learning method for classification and regression that operate by constructing a multitude of decision trees at training time. The ‘forest’ generated by the random forest algorithm is trained through bagging or bootstrap aggregating. The prediction from all trees is majority voted to get the final output during classification. Random Forests are also good at handling large datasets with high dimensionality and heterogeneous feature types. A few hyperparameters were tuned to get the optimum performance based on a CIC-IDS2017 dataset or KDD99 dataset.

Below are the performance metric achieved with random forest model.

CIC-IDS2017

PRECISION		RECALL		F1 SCORE		ACCURACY	FPR	FNR
Attack	Benign	Attack	Benign	Attack	Benign			
0.9844	0.9989	0.9938	0.9973	0.9891	0.9981	0.9968	0.002653	0.006133

***THE DETAILS OF THE KDD CUP 99 DATASET AND PERFORMANCE IS MENTIONED IN APPENDIX-A**

DATASETS FOR DDoS ATTACK

There are a multitude of open-source academia network intrusion datasets available online. Two of the most well-known datasets, namely the KDD Cup 1999 dataset and CIC-IDS2017 dataset, are used in the DDoS intrusion detection experiment detailed in this paper. The KDD Cup 1999 dataset is from the University of California and is the dataset used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99, The Fifth International Conference on Knowledge Discovery and Data Mining. The competition task was to build a network intrusion machine learning model to differentiate between attacks and normal connections.[1]

The CIC-IDS2017 dataset is from the Canadian Institute for Cybersecurity, University of New Brunswick, and it contains benign and more up-to-date common network intrusion attacks, as detailed in Footnote [2]. A more descriptive outline of the details of the dataset and its underlying principles can be found in Footnote [3]. This is an academic intrusion detection dataset aimed specifically at detecting Denial of Service attacks, in particular those which abuse mechanisms at OSI Layer 7. The most well-known example is slowloris which used to eat up all available connections on Apache web servers and then keep them open while pretending to be the slowest client possible.[2]

TRAINING THE RF MODEL

Random forest is an ensemble-based supervised machine learning algorithm that can be used in classification and regression problems. Architecturally, random forest is formed based on aggregation of multiple individually constructed decision trees. In the case of classifying a DDoS attack, the collective predictions generated from all decision trees will be majority voted to obtain the final model classification output categorizing as either benign or attack class for the test data sample in questioned.

For random forest classifier, there are several important model parameters that should be carefully evaluated for optimal model performance. A list of all parameters that are used for model training can be found in Footnote[4]. And some of the more important training parameters that should be carefully chosen are listed below:

- **n_estimators:** This parameter dictates the number of decision trees that will be constructed in the random forest.
- **max_depth:** This parameter is used to control the maximum depth of the trees within the random forest.
- **min_samples_split:** This parameter is used to indicate the minimum number of samples required to split an internal node of the decision trees within the random forest.

In the case of our experimental results for DDoS intrusion detection, the n_estimators, max_depth and min_samples_split parameters were chosen to have the following values as tabulated in Table 1 below for CIC-IDS2017 and KDD Cup 1999 dataset respectively. The selection of the model parameters was determined through grid search approach, in that we observed that training of the random forest classifier with the model parameter values tabulated in Table 1 resulted in the optimal test dataset results for the CIC-IDS2017 and KDD Cup 1999 dataset. It should also be noted that in the process of selecting the optimal model parameter settings as tabulated in Table 1, hardware implementation complexity aspect of the random forest classifier was also taken into consideration, where, in general, it is preferable to have a random forest classifier achieving good model performance with the lowest number of trees (i.e. lowest n_estimators setting) and shallowest tree depth (i.e. lowest max_depth setting).

Table 1: Model parameter settings for CIC-IDS2017 and KDD cup 99 Dataset

	n_estimators	max_depth	min_samples_split
CIC-IDS2017 Dataset	180	8	1000
KDD Cup 1999 Dataset	160	8	1000

**THE DETAILS OF THE KDD CUP 99 DATASET AND PERFORMANCE IS LISTED IN APPENDIX-A*

FEATURE SET (PARAMETERS) TO IDENTIFY THE ATTACKS

In addition, feature selection experiments were also performed to determine a viable subset of reduced feature set that can generate good model performance. The motivation behind the reduced feature list lies in the fact that the lower the number of features required for training and testing of the model, the less complex hardware architecture will be required in the real time feature extraction hardware FPGA implementation of the random forest. To achieve this aim of feature reduction, feature importance and permutation importance analyses were performed on the KDD Cup 1999 Dataset and CIC-IDS2017 Dataset respectively. The reduced feature set for CIC-IDS2017 Dataset is as tabulated in Table 2 below. For completeness, the reduced feature set for KDD Cup 1999 Dataset is also tabulated in Table 3 in the Appendix section.

Table 2: Top features in the order from most- to least-important for CIC-IDS2017 Dataset

FEATURE NAMES	DESCRIPTIONS
FwdHeaderLength.1	Total bytes used for headers in the forward direction
Init_Win_bytes_forward	The total number of bytes sent in initial window in the forward direction
Init_Win_bytes_backward	The total number of bytes sent in initial window in the backward direction
IdleMin	Minimum time a flow was idle before becoming active
AvgBwdSegmentSize	Average size observed in the backward direction
FwdPackets/s	Number of forward packets per second
BwdIATStd	Standard deviation time between two packets sent in the backward direction
BwdIATMax	Maximum time between two packets sent in the backward direction
BwdPacketLengthMin	Minimum size of packet in backward direction
FwdPacketLengthStd	Standard deviation size of packet in forward direction
BwdHeaderLength	Total bytes used for headers in the backward direction
SubflowBwdBytes	The average number of bytes in a sub flow in the backward direction
BwdPackets/s	Number of backward packets per second
SubflowFwdBytes	The average number of bytes in a sub flow in the forward direction
PacketLengthVariance	Variance length of a packet
FwdPacketLengthMin	Minimum size of packet in forward direction
FwdIATTotal	Total time between two packets sent in the forward direction
MinPacketLength	Minimum length of a packet
FwdIATMean	Mean time between two packets sent in the forward direction

**THE FEATURE SET FOR KDD99 DATASETS ARE MENTIONED IN APPENDIX-A*

IMPLEMENTATION OF RANDOM FORREST IN PROGRAMMABLE ACCELERATOR – VERSAL ADAPTIVE SoC

Random forest IP which can be implemented in programmable logic of AMD Versal devices is a parameterizable for the number of trees and depth.

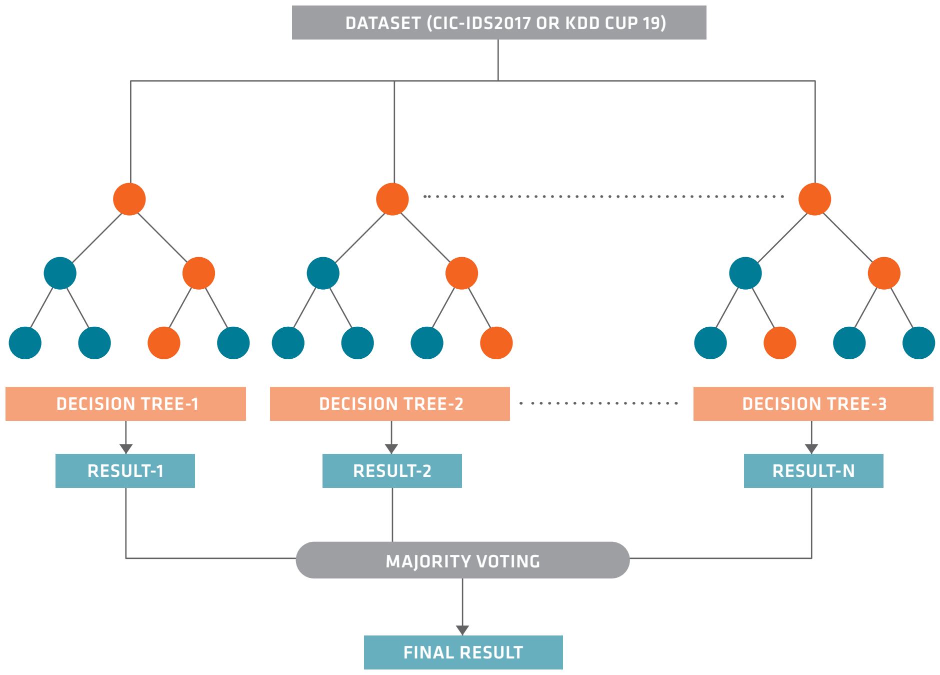
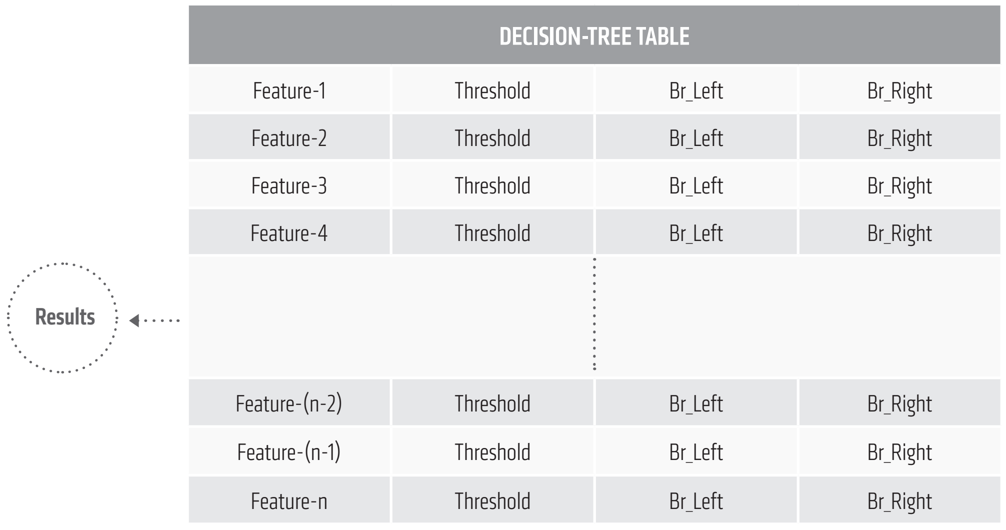


FIGURE 1: FPGA IMPLEMENTATION OF RANDOM FOREST - TRAINED WITH CIC-IDS2017 DATASET

Decision-tree is implemented as a decision logic and a decision-tree table. Addresses to the next node is stored in the decision-tree table. When a decision is made at a node, it branches to the next node pointed to in the decision-tree table.



DECISION-TREE TABLE			
Feature-1	Threshold	Br_Left	Br_Right
Feature-2	Threshold	Br_Left	Br_Right
Feature-3	Threshold	Br_Left	Br_Right
Feature-4	Threshold	Br_Left	Br_Right
⋮	⋮	⋮	⋮
Feature-(n-2)	Threshold	Br_Left	Br_Right
Feature-(n-1)	Threshold	Br_Left	Br_Right
Feature-n	Threshold	Br_Left	Br_Right

FIGURE 2: RANDOM FOREST BRANCHING AND MAJORITY VOTING METHODOLOGY

VERIFICATION OVERVIEW

The random forest IP is pre-verified. However, the test harness design that will be used in validation of the trained RF IP is not pre-verified with the used datasets. To mitigate this gap and to minimize any on board debug encountered during the validation on reference platform, we have developed a simulation- based verification testbench.

NIAD CoCo TB Python Simulation Setup

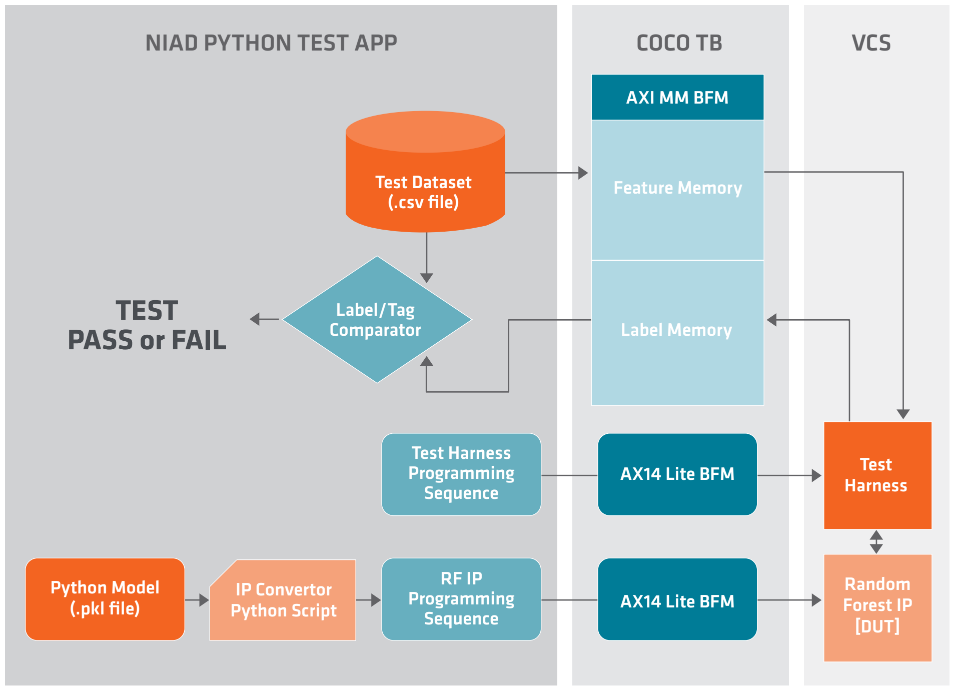


FIGURE 3: VERIFICATION ARCHITECTURE FOR FPGA-BASED RANDOM FOREST IMPLEMENTATION.

Since the collateral from the design and IP team would be easy to be consumed in Python, the testbench is built using the CoCoTB Python package. After the training of the RF model is done, the parameters of the trained RF model are stored in a pickle file. The testbench converts this pickle file into its corresponding RTL configuration registers. The testbench programs the IP by writing the configuration registers using its AXI4 Lite interface. Based on the test setting, the testbench then picks either a portion of the dataset or the complete dataset for testing the IP. It packs the feature columns of the chosen vectors as required by the test-harness design and loads them into the AXI memory. It then configures the memory locations of the test harness where the features are stored and the location where the classification results need to be stored. The testbench then triggers the test-harness to start its operation and polls for the TEST-DONE status bit to be set. Once the test is done, the testbench reads back the results and compares it against the expected results from the dataset. The testbench also derives the run's performance measurement using the test-harness' status registers.

VALIDATION ON AMD VERSAL PREMIUM

The random forest IP is implemented on the VPK120 reference platform. The VPK120 Evaluation Kit offers networked, power-optimized cores paired with many high-speed connectivity options. The kit features the Versal Premium VP1202 adaptive SoC, which integrates 100+Gb/s PAM4 transceivers, PCIe® Gen5 with DMA & CCIX, 100G multirate Ethernet cores, 600G Ethernet cores, 400G High-Speed Crypto Engines, and more.

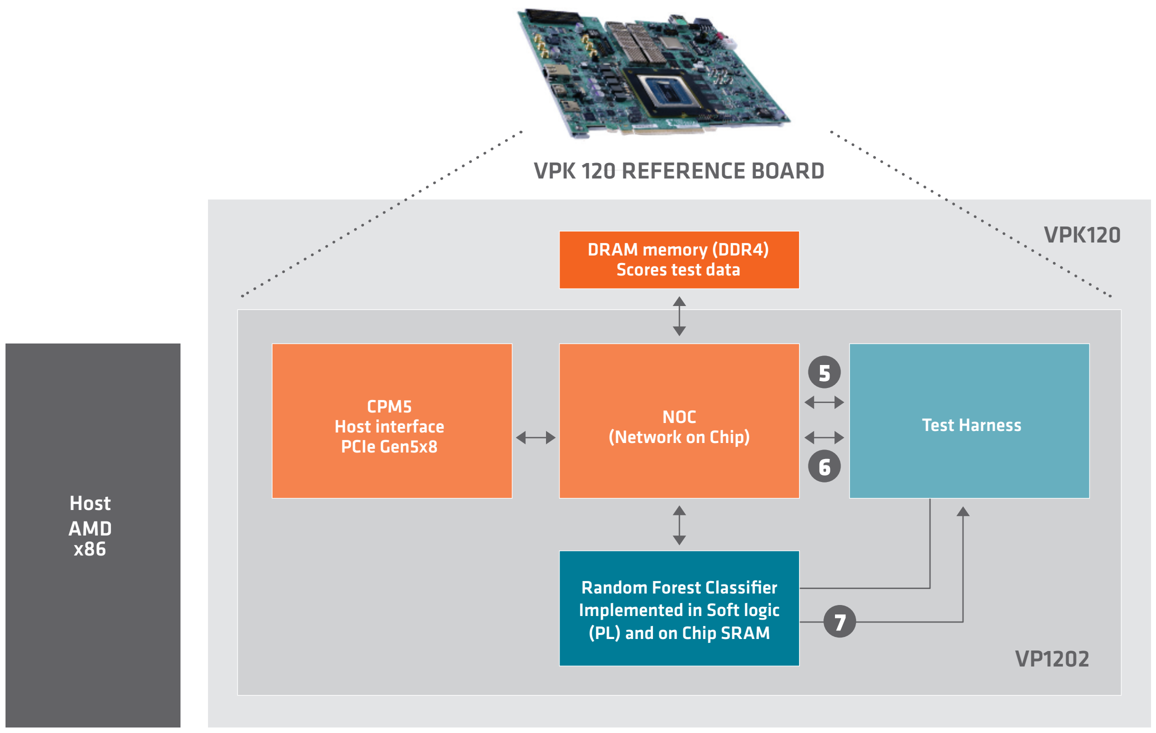

VPK 120 REFERENCE BOARD
VPK120
DRAM memory (DDR4) Scores test data
Host AMD x86
CPM5 Host interface PCIe Gen5x8
NOC (Network on Chip)
5
6
Test Harness
Random Forest Classifier Implemented in Soft logic (PL) and on Chip SRAM
7
VP1202

FIGURE 4: HIGH-LEVEL DESIGN ARCHITECTURE FOR RANDOM FOREST IP

	I/F STD.	DESCRIPTION
1	PCIe	Host downloads test data, uploads inference results, program nodes configuration, and access test-harness registers.
2	AXI4-MM	Write test data, read inference result, program nodes configuration, and access test-harness registers.
3	AXI4-Lite	program nodes configuration
4	LPDDR4	Write test data, read inference result
5	AXI4-Lite	Test-harness registers read write
6	AXI4-MM	Read test data, write inference result
7	AXI4-Stream	Test data and inference result

RANDOM FOREST IO HAS 2X AXI4-STREAM INTERFACES (INPUT AND OUTPUT) AND 1X AXI4-LITE INTERFACE.

The validation platform uses CPM5 for interfacing the VPK120 to the host system via PCIe. The host configures random forest module, loads test data, reads inferred results, and controls test-harness registers via CPM5.

8GB of LPDDR4 on VPK120 is partitioned into two segments for test data and inference results.

The test-harness fetches test data from LPDDR4 and inputs them to the random forest module. It receives inferred results from the random forest module and writes them to LPDDR4.

Through the test-harness' registers the host can,

1. Indicate test data size,
2. Start test,
3. Poll for test completion
4. Check elapsed clock cycles from start to completion of test.

PERFORMANCE WITH HARDWARE IMPLEMENTATION

The random forest IP is a parameterized synthesizable RTL representation of the random forest algorithm.
The random forest IP on Versal is used for inference only as the model is trained offline usin the CPU.
The random forest IP is implemented in Versal Premium with the following parameters.

- Number of trees = 200
- Maximum number of nodes= 255
- Maximum feature vector size= 255
- 32-bit floating point enabled.

Logic utilization in the Versal Premium device XCVP1202-VSVA2785 is as follows.

LUT		FF		BRAM36	
44861	4.98%	55357	3.07%	300.5	22.40%

THE INFERENCE/SECOND FOR KDD99 AND CIC-IDS2017 DATASET IS APPROX. 6 MILLION INFERENCES PER SECOND.

KDD99 and CIC-IDS2017 models were run on an AMD Ryzen™ 7 PRO 6850U laptop processor and approximately 600K-700K inferences per second were achieved using all available eight cores. Though there are many different options among the available CPU core classes in terms of different cache size, operating frequency and available DRAM memory, the FPGA implementation of the random forest algorithm provides us some rough estimates of the expected performance boost resulted by using an adaptive hardware accelerator.

RF Classification Performance

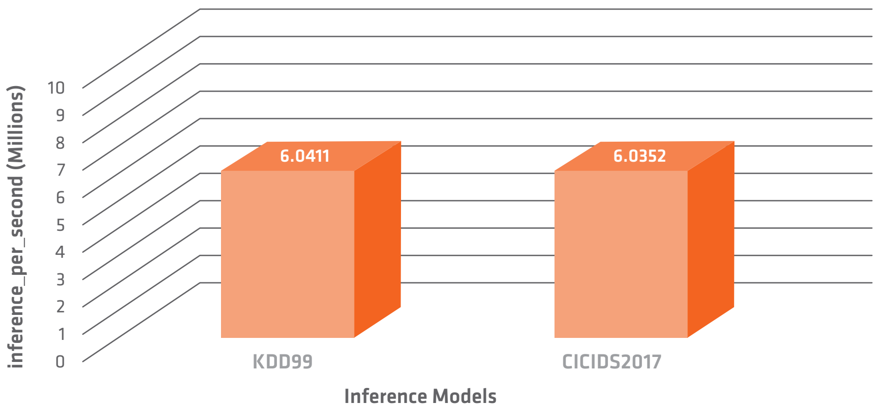


FIGURE 5: RANDOM FOREST PERFORMANCE FOR FPGA IMPLEMENTATION

EXTENSION TO OTHER MODELS

The deep neural network models are developed for the CIC-IDS2017 dataset. AMD FPGA offerings have devices with AI Engines. The AI Engines are dedicated vector engines on Versal devices that are optimized for neural network architecture implementation. The other type of network models such as encoder/decoder are also being explored as they have shown significant performance and accuracy in anomaly detection.

SUMMARY

Random forest classifiers are scalable, lightweight AI models which are suitable for implementation in programmable logic. In this implementation we have optimized the number of trees and number of nodes to minimize the FPGA resource count. The results showed that a trained model is still able to generate the high accuracy (> 0.99) with both openly available dataset KDD99 and CIC-IDS2017. Since the feature set for the DDoS traffic can be large, we may need a higher number of tree and node counts in some cases which might affect the FPGA resources.

This is a lookaside implementation where traffic feature extraction is done in software and extracted traffic parameters are sent to the FPGA via the PCIe interface to be analyzed by a machine-learning model running on an FPGA-based accelerator. Since FPGAs are well-suited and used as packet/traffic and flow classifiers in network firewalls, we plan to extend the role of the FPGA as an in-line flow processor where the network traffic flows are extracted, and traffic features are made available to the random forest model without the CPU assistance. This in-line implementation can have significant performance advantages in terms of throughput and latency.

Another extension of the implementation is planned as replacement of the random forest model to a deep-learning model with multiple layers. The use of deep neural network models may bring significant advantages while using a large number of traffic features for the DDoS mitigation.

APPENDIX A

Table 3: Top features in the order from most- to least-important for KDD Cup 1999 Dataset

FEATURE NAMES	DESCRIPTIONS
src_bytes	Number of data bytes transferred from source to destination in single connection
dst_bytes	Number of data bytes transferred from destination to source in single connection
serror_rate	The percentage of connections that have activated the flag s0, s1, s2 or s3, among the connections aggregated in count
flag	Status of the connection – Normal or Error
service	Destination network service used
dst_host_rerror_rate	The percentage of connections that have activated the flag REJ, among the connections aggregated in dst_host_count
dst_host_serror_rate	The percentage of connections that have activated the flag s0, s1, s2 or s3, among the connections aggregated in dst_host_count
logged_in	Login Status : 1 if successfully logged in; 0 otherwise
rerror_rate	The percentage of connections that have activated the flag REJ, among the connections aggregated in count
diff_srv_rate	The percentage of connections that were to different services, among the connections aggregated in count
dst_host_same_src_port_rate	The percentage of connections that were to the same source port, among the connections aggregated in dst_host_srv_count
dst_host_srv_count	Number of connections having the same port number
dst_host_srv_rerror_rate	The percentage of connections that have activated the flag REJ, among the connections aggregated in dst_host_srv_count
dst_host_same_srv_rate	The percentage of connections that were to different services, among the connections aggregated in dst_host_count
same_srv_rate	The percentage of connections that were to the same service, among the connections aggregated in count
dst_host_srv_serror_rate	The percent of connections that have activated the flag s0, s1, s2 or s3, among the connections aggregated in dst_host_srv_count
dst_host_diff_srv_rate	The percentage of connections that were to different services, among the connections aggregated in dst_host_count
srv_serror_rate	The percentage of connections that have activated the flag s0, s1, s2 or s3, among the connections aggregated in srv_count
count	Number of connections to the same destination host as the current connection in the past two seconds

KDD Cup 99 Performance

PRECISION		RECALL		F1 SCORE		ACCURACY	FPR	FNR
Attack	Benign	Attack	Benign	Attack	Benign			
0.9927	0.9992	0.9983	0.9967	0.9955	0.9979	0.9972	0.003318	0.001749

REFERENCES

1. University of California, 1999, “The UCI KDD Archive,” University of California, Department of Information and Computer Science. [Online]. Available: http://kdd.ics.uci.edu
2. https://www.unb.ca/cic/datasets/ids-2017.html
3. I. Sharafaldin, A. Lashkari, and A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization,” in *Proc. 4th Int. Conf. Inform. Syst. Secur. Priv. (ICISSP),* 2018, pp. 108-116.
4. F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,” J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.

For any questions on implementation and related files, please contact one of the following:

AUTHORS

Awanish Verma (awanishv@amd.com)

Rajesh Bansal (rajesh.bansal@amd.com)

Sukruth Pattanagiri (sukruth.pattanagiri@amd.com)

Yew Kwan Chong (yew.kwan.chong@amd.com)

Huang, Chee Cheun (chee.cheun.huang@amd.com)

AMD.com/workstation