Received September 5, 2020, accepted September 21, 2020, date of publication September 25, 2020, date of current version October 6, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3026831

FPGA-Based Network Traffic Classification Using Machine Learning

MOHAMMED ELNAWAWY, ASSIM SAGAHYROON, (Senior Member, IEEE), AND TAMER SHANABLEH, (Senior Member, IEEE)
Department of Computer Science and Engineering, American University of Sharjah, Sharjah, United Arab Emirates

Corresponding author: Mohammed Elnawawy (melnawawy@aus.edu)

This work was supported in part by the Department of Computer Science and Engineering, American University of Sharjah, and in part by the Open Access Program from the American University of Sharjah.

ABSTRACT Real-time classification of internet traffic is critical for the efficient management of networks. Classification approaches based on machine learning techniques have shown promising results with high levels of accuracy. In this article, the suitability of packet-level and flow-level features is validated using stepwise regression and random forest feature selection. Moreover, the optimal percentage of packets considered within a flow while extracting flow-level features is determined. Several experiments are conducted using naïve Bayes, support vector machine, k-nearest neighbor, random forest, and artificial neural networks on the University of Brescia (UNIBS) and the University of New Brunswick (UNB) datasets, which are both publicly available. The performed experiments show that 60% of flow packets are a good compromise that ensures high performance in the least processing time. The results of the conducted experiments indicate that random forest outperforms other algorithms achieving a maximum accuracy of 98.5% and an F-score of 0.932. Further, and since software-based classifiers cannot meet the anticipated real-time requirements, we propose a Field-Programmable Gate Array (FPGA) based random forest implementation that utilizes a highly pipelined architecture to accelerate such a time-consuming task. The proposed design achieves an average throughput of 163.24 Gbps, exceeding throughputs of reported hardware-based classifiers that use comparable approaches, which in turn ensures the continuity of real-time traffic classification at congested data centers.

INDEX TERMS Feature extraction, FPGA, machine learning, random forest, traffic classification.

I. INTRODUCTION

The Internet has been one of the most important inventions of the twentieth century. To cope with the increasing number of internet users, companies are constantly working on enhancing their internet speeds. The prosperity of the internet and its growing speeds has allowed more traffic to flow in and out of the average computing device. However, it also opens doors for potential threats and malicious attacks. Therefore, researchers have realized the need for proposing several traffic classification techniques that help manage and control the flow of network traffic to alleviate the risks involved with potential threats.

Traffic classification is the association of network traffic with the application or category of applications that generated them (for example, Skype, HTTP, SMTP, video streaming, and so on). Traffic classification is important for several reasons [1], namely, ensuring the Quality of Service (QoS) and Service Level Agreement (SLA) and troubleshooting abnormal network behavior during unexpected downtimes whereby network administrators could potentially use it to identify points of failure within the network. Traffic classification is also used for traffic shaping and bandwidth allocation which regulates the flow of network packets to ensure compliance with a specific traffic profile. Lastly, traffic classification is used in cyber-security since it helps recognize malicious classes of traffic that include viruses, trojans, spyware, and many others. Once a specific flow of traffic has been labeled as malicious, an Intrusion Detection System (IDS) can then block out the malicious classes before they reach the user.

Traffic classification techniques are divided into four main mechanisms; port-based, Deep Packet Inspection (DPI) based, heuristic-based, and Machine Learning (ML) based

The associate editor coordinating the review of this manuscript and approving it for publication was Li Zhang.

techniques [2]. Port-based classification techniques are largely reliant on the port numbers of the transport layer of the Open System Interconnection (OSI). However, as communication protocols evolve, applications started varying their port numbers dynamically to obfuscate any means of traffic classification. DPI focuses on invasively checking the payload of the traffic looking for known signatures that relate to specific applications to categorize it. Nevertheless, it is also one of the most time and resource consuming techniques since pattern matching on application signatures requires lots of computing power besides the time required to compare a signature to a database of pre-saved signatures for classification purposes. Also, some people are worried about the privacy of their communicated data since they do not wish to be monitored. Consequently, applications started overcoming this mechanism by encrypting the payloads of their packets to protect their contents. Accordingly, encryption renders DPI completely impractical. Furthermore, heuristic techniques tend to consume lesser resources, produce the output in a shorter time at the expense of sacrificing the classification quality since it results in very low accuracies [3].

On the other hand, machine learning classifiers do not need to know the content of a packet to be able to classify it, rather, the classifier can classify traffic traces by making use of their important statistical information. The statistical information consists mainly of flow-level features like packet interarrival times, average size, maximum size, and many more [4]. A flow is defined as a series of packets that share the same source and destination IP addresses, source and destination port numbers, and protocol. Moreover, machine learning techniques tend to offer a greater deal of quality results when compared to heuristic techniques [3]. In Section II, the different machine learning approaches taken by researchers to implement a traffic classifier are investigated.

II. RELATED WORK

In this section, existing work within the traffic classification field is examined to study the progress of research within this area and pinpoint potential room for advancement.

A. SOFTWARE-BASED TRAFFIC CLASSIFIERS

Software-based implementations of traffic classifiers rely on software programs that are executed on general-purpose microprocessors. The authors in [5] used two datasets which consist of 14 different classes including Skype, FileZilla, Facebook, Torrent, Twitter, and many more. In their work, the authors were able to extract an initial set of 111 features. WEKA was then used to test four different algorithms, namely J48, Random Forest, k-Nearest Neighbor (KNN), and Bayes Net. The results of their experiments show that KNN ($k = 1$) and Random Forest have the best performances with accuracies of 93.94% and 93.74% respectively.

Another study proposed a real-time Support Vector Machine (SVM) traffic classifier that was implemented using the CoMo project infrastructure [6]. They select eight different classes to work with, which are web browsing, peer-to-peer, DNS, email, network operation, encrypted traffic, chat, and attacks. Each class is modeled as a test function that is used to determine the probability of an instance belonging to this class. The results obtained from their experiments show that the system can work at speeds reaching up to 600 Mbps. A drawback of such a software system is its inefficiency in handling high throughputs. This becomes a serious problem at data centers and Internet Service Providers (ISPs) operating at 10s and 100s of Gbps [3]. This limitation paves the way for the introduction of hardware accelerators like GPUs and FPGAs that can be used to perform the computationally intensive machine learning operations at speeds of 100s of Gbps.

B. HARDWARE-BASED TRAFFIC CLASSIFIERS

Unlike software classifiers, hardware-based implementations rely on dedicated hardware designs that are used to speed up the classification process. The hardware traffic classifiers are usually implemented using FPGA or GPU-based designs. The system proposed in [7] suggests using eight candidate features that describe traffic traces. A C4.5 decision tree algorithm is used which resulted in the best performance according to their literature review. The authors suggest the use of two algorithms when implementing the C4.5 decision tree on the FPGA, namely, Optimized Decision Tree (ODT) and Divide and Conquer (DQ). To further optimize those algorithms for FPGA implementation, the authors decide to use pipelined architectures for both ODT and DQ whereby at each clock cycle one input is consumed and one output is generated. However, the authors did not compare their achieved throughput to that of software.

The study in [8] suggests a different approach to traffic classification which uses SVM on an FPGA accelerator. In contrast to the previous paper, the authors demonstrate both a software and a hardware implementation of their algorithm using NetFPGA 10G FPGAs that incorporate four network interfaces at 10 Gbps each. A potential weakness of their work is the fact that the generated SVM model was stored inside a Read-Only Memory (ROM) instead of the faster Random-Access Memory (RAM). This entails incurring a long time to reconfigure the FPGA in case the SVM model needed to be retrained when new application types show up in the network.

Hardware platforms such as FPGAs are used mainly because of their advantage in accelerating the computation of certain complex tasks. They are known to be faster than software at intricate computations since they avoid the overhead of unnecessary software calls. Hardware is also faster because special architectures can be designed to perform a certain task more efficiently at the physical layer avoiding the need to go through the time and resource-consuming process of translating the high-level code into machine language. FPGAs' internal architecture supports true parallelization which enables real parallel execution of instructions that facilitates a massive improvement to classification speed compared to software-based classifiers as well as traditional hardware-based ones.

Nevertheless, FPGAs tend to suffer from some limitations including the fact that it is much easier to design a software-based traffic classifier as opposed to an FPGA-based one. This is because it is usually very complex to program an FPGA due to its parallel architecture, as well as the use of hardware description languages like Verilog Hardware Description Language (HDL) which tend to be difficult [9]. Also, FPGAs may consume more power compared to special low-power architected microprocessors or microcontrollers. Therefore, in data centers where power consumption is critical the use of FPGAs must be of clear advantage.

III. PROBLEM STATEMENT

The reviewed work did not consider a systematic approach towards choosing the optimal number of packets to be considered within a flow to extract flow-level features, rather, the first n packets within a flow are used. Large values of n enhance the classification at the expense of increasing the delay, whereas small values of n result in poor classification accuracies. In this work, we find n that strikes a compromise between classification and delay. Moreover, additional flow-level features are considered to study their effect on classification performance. Besides, most of the literature in this field attempt to build classifiers that make use of the port numbers which are becoming an obsolete way of classifying traffic since applications try to dynamically disguise their ports to obfuscate any means of traffic classification. Therefore, an efficient classifier that does not depend on port numbers is devised in this article.

The implementations of software-based traffic classifiers do not cope with the enormous amount of network traffic and hence the need for hardware-accelerated traffic classifiers. In the literature, almost all papers did not study and analyze hardware-implemented random forest network traffic classifiers. One of the main features of the random forest algorithm is its ability to generalize well and avoid overfitting to the training set when compared to single decision trees like C4.5. Therefore, in this article, a hardware-based network traffic classifier using random forests on FPGA is proposed.

The contributions of this article can be summarized as follows:

1. The introduction of a novel set of important features that rely on entropy to effectively classify network packets.
2. The reviewed literature does not provide any means of scientifically selecting the value of n for the first n packets in a flow that must be considered when extracting flow-level features. In this work, the percentage of packets within a flow that strikes a balance between high classification performance and short waiting time before the packets arrive for feature extraction is methodically determined.
3. To the best of our knowledge, we introduced a hardware-based pipelined architecture of a random forest network traffic classifier that has not been investigated in the literature before. Its superiority and usefulness in terms of accuracy and throughput are shown compared to other machine learning algorithms in time-critical domains such as real-time traffic classification.
4. At the hardware implementation level, a novel idea known as the ‘‘effective address’’ is introduced at each tree level. Effective Address (EA) helps eliminate the need to duplicate node information at every tree level; hence reducing the memory requirements of the random forest implementation.

IV. DATASETS

This section discusses the two different datasets that were used in our experiments, the UNIBS and the UNB datasets.

A. THE UNIVERSITY OF BRESCIA DATASET

The dataset in [10], [11] contains several traffic traces and their associated ground truth information collected using the Ground Truth (GT) tool [12], which is a tool developed in 2009 to obtain the ground truth of a traffic trace. The traffic traces were captured at the edge router of the university's campus network on three consecutive days. Twenty workstations running the GT tool were used to collect the traces. As a result, the UNIBS dataset was captured in a non-controlled environment whereby all types of traffic could flow into the system with almost no restrictions, and then GT was used to obtain the ground truth of the captured packets. The UNIBS dataset resulted in traffic traces of around 27 GB of data. The traffic classes used from the UNIBS dataset in this work are Browser, Mail, RSS feed, BitTorrent, and Skype. The number of packets used from the UNIBS dataset is 45541 packets of which 22249 belong to Browser, 1291 belong to Mail, 279 belong to RSS feed, 946 belong to BitTorrent, and 20776 belong to Skype.

B. THE UNIVERSITY OF NEW BRUNSWICK DATASET

The UNB dataset [13] was captured in 2016 which makes it quite recent compared to the 2009 UNIBS dataset. The collection of the UNB traces was conducted in a controlled environment which ensured that all services except the target service were shut down before the collection process started. In capturing the traffic traces, the researchers used packet analyzers like Wireshark and tcpdump which resulted in total traffic of 28 GB of data. The traffic classes used from the UNB dataset in this work are YouTube, Netflix, Spotify, Torrent, and Skype. The number of packets used from the UNB dataset is 99818 packets of which 19996 belong to YouTube, 20000 belong to Netflix, 19885 belong to Spotify, 19986 belong to Torrent, and 19951 belong to Skype.

V. PROPOSED FEATURE EXTRACTION AND SELECTION

In feature extraction, packet-level and flow-level features are extracted from both datasets [4], [14]. Table 1 shows a summary of all the 26 extracted features. The flow-level features that use entropy were not previously discussed

TABLE 1. Complete list of extracted features.

Packet-Level Features	Flow-Level Features	
Source port	Minimum frame length	Variance capture length
Destination port	Maximum frame length	Entropy capture length
Protocol	Mean frame length	Flow size capture length
Time to live	Median frame length	Minimum interarrival time
	Variance frame length	Maximum interarrival time
	Entropy frame length	Mean interarrival time
	Flow size frame length	Median interarrival time
	Minimum capture length	Variance interarrival time
	Maximum capture length	Entropy interarrival time
	Mean capture length	Number of packets in flow
	Median capture length	Flow duration

in the literature. Entropy refers to the average number of bits needed to represent the outcome of the experiment. It is defined as $-\sum p_i \log_2(p_i)$, where p_i is the probability of occurrence of the event i.

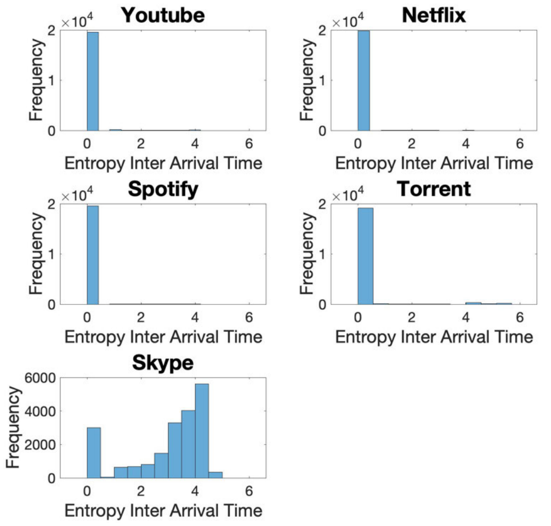

FIGURE 1. UNB entropy interarrival time histogram.

The histogram of each feature per traffic class is plotted. For example, the entropy interarrival time of the UNB dataset is shown in Fig. 1. The figure highlights the different distribution of the UNB entropy interarrival time for the Skype class compared to all other classes. This figure helps in the preliminary analysis which anticipates that entropy interarrival time will be retained as an important feature after performing feature selection. Such a feature is useful in distinguishing the Skype traffic from the rest since it is distinct from the other classes.

On the other hand, the histograms of other features showed that they might not be very useful in differentiating classes due to the overlap between their values. Fig. 2 shows the histogram of the flow duration of the UNIBS dataset.

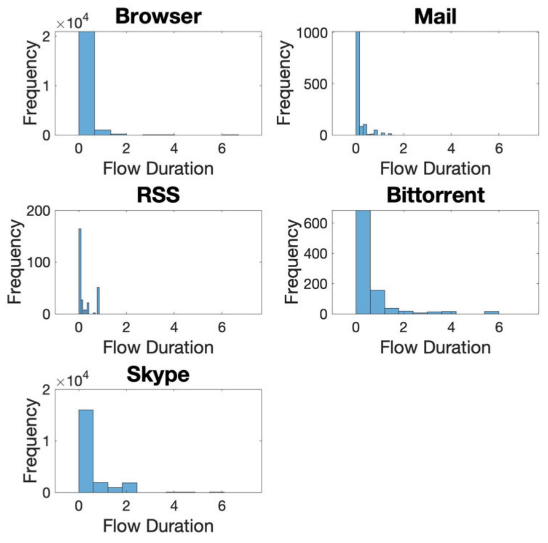

FIGURE 2. UNIBS flow duration histogram.

The figure shows very similar distributions of UNIBS flow duration among the five different classes which suggests that the feature might be discarded through feature selection as it does not differentiate well between the different traffic classes. Therefore, these histograms help in gaining a good first insight into the distribution of the different features and their importance to the classification performance. Nevertheless, it is difficult to solely rely on these histograms to determine what features to retain and what features to discard. Hence, feature selection algorithms are used to identify the most important features.

In this work, two feature selection algorithms are used to retain the most important features among the 26 extracted features, namely, Stepwise Regression (SWR) [15] with a minimum p-value of 0.025 and Random Forest (RF) feature selection with a threshold of 90%.

Table 2 and Table 3 show the selected features using stepwise regression from the UNIBS and UNB datasets, respectively, while Table 4 and Table 5 show the selected features using random forest from the UNIBS and UNB datasets, respectively.

It is interesting to see that the entropy of interarrival time is retained by both algorithms whilst the flow duration is discarded by both. This reassures the initial analysis of the histograms presented in Fig. 1 and Fig. 2.

VI. SOFTWARE IMPLEMENTATION

In the proposed software implementation of the traffic classifier, a 10-fold cross-validation experiment is performed on seven different subsets of features shown in Table 6 to investigate the effect of using each combination of features on the performance of the classifiers. Five different classifiers are built, namely, naïve Bayes, linear SVM, second-order polynomial SVM, KNN, and random forest. To ensure fairness when comparing the different models, the training and testing process was done on the same training and testing sets.

TABLE 2. UNIBS features retained by stepwise regression.

Packet-Level Features	Flow-Level Features	
Source port	Minimum frame length	Mean capture length
Destination port	Maximum frame length	Median capture length
Protocol	Mean frame length	Variance capture length
Time to live	Median frame length	Entropy capture length
	Variance frame length	Minimum interarrival time
	Entropy frame length	Maximum interarrival time
	Minimum capture length	Variance interarrival time
	Maximum capture length	Entropy interarrival time

TABLE 3. UNB features retained by stepwise regression.

Packet-Level Features	Flow-Level Features	
Source port	Minimum frame length	Variance frame length
Destination port	Maximum frame length	Entropy frame length
Protocol	Mean frame length	Minimum interarrival time
Time to live	Median frame length	Entropy interarrival time

TABLE 4. UNIBS features retained by random forest.

Packet-Level Features	Flow-Level Features	
Source port	Maximum frame length	Median capture length
Destination port	Mean frame length	Flow size capture length
Protocol	Median frame length	Minimum interarrival time
Time to live	Variance frame length	Maximum interarrival time
	Entropy frame length	Mean interarrival time
	Minimum capture length	Median interarrival time
	Maximum capture length	Variance interarrival time
	Mean capture length	Entropy interarrival time

That is, the same seed in random number generators was used to recreate the same environment and ensure the comparison is fair among all algorithms.

The port-less experiments are very important since, in the worst-case scenario, if port numbers were dynamically changed by the different applications, they will no longer influence the classification process. Therefore, one of the aims of this article is to build classifiers that could counteract the obfuscation process and still be able to classify traffic without the need to have fixed port numbers per application.

Also, the software system is required to wait for a number of packets within a flow to extract flow-level features. In this

TABLE 5. UNB features retained by random forest.

Packet-Level Features	Flow-Level Features	
Source port	Minimum frame length	Maximum capture length
Destination port	Maximum frame length	Mean capture length
Protocol	Mean frame length	Median capture length
Time to live	Median frame length	Variance capture length
	Variance frame length	Flow size capture length
	Entropy frame length	Median interarrival time
	Flow size frame length	Entropy interarrival time
	Minimum capture length	

TABLE 6. Feature subsets.

	Subset Name	Subset Features
1 -	All Features	All 26 extracted features
2 -	All Features Without Port Numbers	Similar to subset 1 but source and destination ports are excluded
3 -	Port Numbers Only	Only source and destination ports
4 -	Stepwise Regression (SWR) Features	Features retained by the SWR algorithm
5 -	Stepwise Regression (SWR) Features Without Port Numbers	Similar to subset 4 but source and destination ports are excluded
6 -	Random Forest (RF) Features	Features retained by the RF algorithm
7 -	Random Forest (RF) Features Without Port Numbers	Similar to subset 6 but source and destination ports are excluded

work, the percentage of packets used to extract flow-level features in every flow is varied such that the packet percentage varies from 10% to 100% of the packets in a flow using the 10-fold cross-validation method. The average wait time to receive the required percentage of packets is also plotted against the percentage of packets used from the traffic flow. As such, the most optimal packet percentage required for flow-level feature extraction is obtained.

To find out the optimal number of trees within the random forest that would lead to the best classification performance, the average out-of-bag error is plotted against the number of trees within the forest. To do so, the number of trees is varied from 1 to 500 for both datasets. Consequently, the optimal number of trees is found to be approximately 50 trees.

VII. HARDWARE IMPLEMENTATION

In the software-based experimental results demonstrated in Section VIII, it is shown that the random forest algorithm tends to outperform all other algorithms in terms of classification F-score. In this work, the macro-averaged F-score is used to calculate the arithmetic mean of the per-class F-scores.

Hence, we choose to design a hardware accelerator based on a random forest classifier using Verilog HDL to speed up the classification process using hardware. Therefore, the highly parallel architecture of FPGAs is exploited to accelerate the design even further.

In a different implementation domain, authors in [16] suggested two possible architectures for a random forest implementation on hardware, memory-centric and comparator-centric. The memory-centric approach enables a quick context switching from one random forest model to another through simply loading new node information into the tree level memory. On the contrary, the comparator-centric approach results in very high consumption of FPGA comparators, while requiring no memory elements to store the node information. Keeping in mind the use of two datasets, this entails the need to incorporate the context switching feature of the memory-centric architecture. Therefore, in this article, and with noticeable modifications to the implementation suggested in [16], the proposed implementation will follow the memory-centric approach.

In this section, the hardware design of the proposed random forest classifier is discussed in detail. The DE2-115 development board manufactured by Terasic Inc. which features a Cyclone IV E FPGA chip designed and manufactured by Altera (now Intel) is used. This work focuses on the random forest classifier core and assumes the existence of an interface between the core and the network module used to receive real-time traffic.

In the proposed implementation, the 26 features are encoded using binary numbers where each feature is encoded as a 58-bit fixed-point number. Fixed-point was chosen instead of floating-point since it usually results in a much simpler hardware design which tends to be faster than a floating-point architecture. Each of the 26 features is encoded such that 30 bits resemble the integer part and 28 bits are used to describe the fractional part of the number. This results in an encoding scheme that requires 1508 bits to describe the 26 features of one network packet. Note that only the features that were retained by random forest feature selection (shown in Table 4 and Table 5), excluding port numbers, were used to build the final random forest model. Encoding the full feature set here simply means accounting for the extraction of the full feature set. Nevertheless, only selected features are used for classification.

The main objective while designing the random forest classifier in hardware is to identify independent components that can work simultaneously without affecting the operation of one another. The most obvious independent components are the individual trees within the forest since a test instance is simply passed down each tree regardless of the output of the other trees. The structure of the independent trees is shown in Fig. 3 which offers an overview of the hardware-based random forest design. The figure shows that the test packet is registered at an input register which introduces a one-cycle delay. Once the test instance reaches the trees it goes into the different levels of the trees. Each tree level will pass on the packet to the next level within the tree for more checks, along with the address of the next node in the tree.

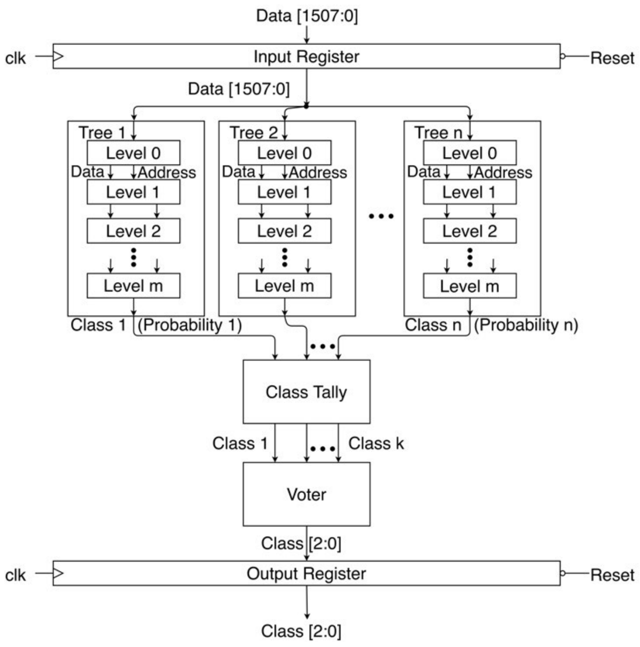

FIGURE 3. **Hardware-based random forest design overview.**

The execution of each level within the tree is also one more aspect that requires attention. Each tree level examines only one packet at a time, therefore, instead of treating the entire tree as one bulky component, pipeline stages are inserted between the different tree levels. This is yet another aspect where the proposed design exploits the parallel execution capabilities of FPGAs since now tree levels can operate simultaneously and independently with respect to all other levels within the tree. The output of level m is either the class label in the case of a majority-based random forest or class probabilities in the case of a probability-based random forest. After that, the outputs of all trees are fed into a module known as "Class Tally." The Class Tally module will simply aggregate the results of all trees and will then pass the results to the "Voter" module. The Voter module will eventually choose the most occurring class (majority-based) or the class with the highest probability (probability-based) to be the class label of the data instance. Lastly, to further add on to the highly pipelined architecture, an output register is used to simply register the output class such that it can be displayed to the user promptly.

One of the contributions of this work is the introduction of the concept of effective address within the proposed design. Effective Address (EA) is the address of a node in its respective tree-level starting from effective address 0 for the first node in level m. The effective address helps eliminate the need to duplicate node information at every tree level; hence reducing the memory requirements of the random forest implementation. The effective address of a node within level m can be calculated using (1), where the original address is the actual node number within the whole tree.

$$\text{EA} = \text{Original Address } - 2^m + 1 \tag{1}$$

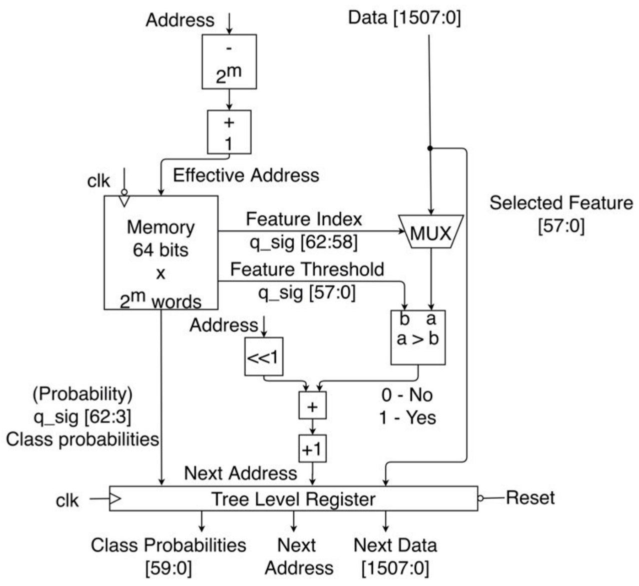

FIGURE 4. Tree level architecture.

Fig. 4 shows the hardware components inside a tree level. Firstly, the effective address is computed using a subtractor and an adder. The effective address is then used to index the tree memory which holds the information about all nodes within the respective tree level. The tree memory will fetch the feature index which is simply the index of the feature being checked within this node. As mentioned in Section V, 26 features have been extracted, so feature index is a number between 0 and 25. The feature index is then used as selection lines to a multiplexer that will enable the value of only the feature under inspection to pass through for comparison. The tree memory will also provide the feature threshold which is the value against which the feature value is compared. The comparator checks whether the selected feature is greater than the feature threshold. The result of the comparison is then used to calculate the next address.

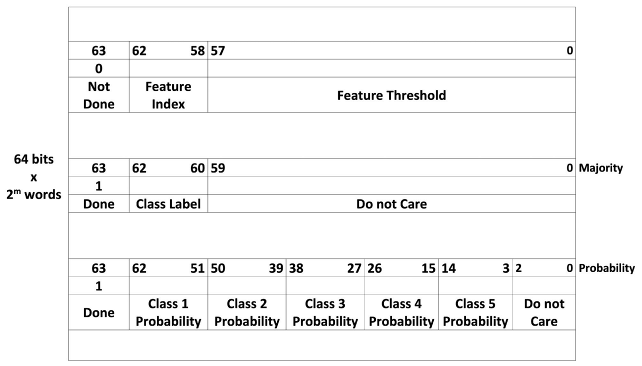

FIGURE 5. Tree memory design.

Fig. 5 gives a better insight into the design of the tree memory and shows how the different fields of a node in the tree are stored in the memory. To exploit the parallel capabilities of the FPGAs, the on-chip RAM is used to act as the tree memory and store all the node information. This is due to the ability of an FPGA to restructure its on-chip memory on-demand such that each tree level can have simultaneous access to its tree-level memory without creating a memory access bottleneck at the other levels of the same tree or even other trees in the forest. By doing so, all levels in all trees can fetch their node information at the same time.

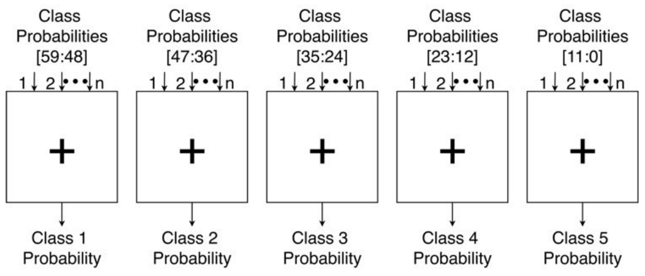
Class Probabilities [59:48]
Class Probabilities [47:36]
Class Probabilities [35:24]
Class Probabilities [23:12]
Class Probabilities [11:0]
1 2 n
+
Class 1 Probability
Class 2 Probability
Class 3 Probability
Class 4 Probability
Class 5 Probability

FIGURE 6. Class tally module design (probability-based).

The Class Tally module of a probability-based algorithm finds the sum of probabilities of each class resulting from the probabilities obtained from each decision tree. Fig. 6 shows that each class has a dedicated adder to calculate the sum of the probabilities of the corresponding class. The output of the Class Tally module is the sum of all probabilities for each class.

Fig. 7 shows the architecture of the Voter module of a probability-based model.

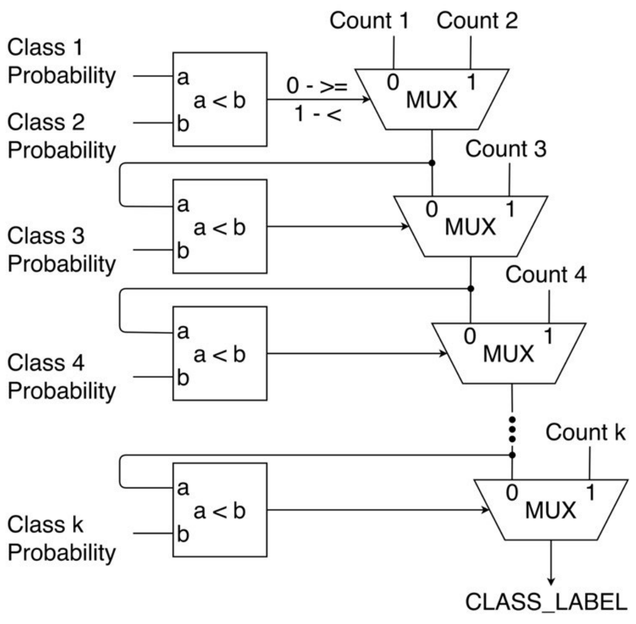

FIGURE 7. Voter module architecture (probability-based).

The Voter module selects the class with the highest sum of probabilities. The k probabilities are simply compared and the class with the highest probability is chosen. The values of class 1 probability and class 2 probability are compared using a comparator. The result of the comparison is used as a selection line to a multiplexer that routes the class with a higher probability for further comparisons. To keep track of which class count was routed through the multiplexer, the CLASS_LABEL, which is a number from 1 to k that represents the class, is concatenated with the class probability corresponding to it. Hence, Count k consists of {CLASS_LABEL, Class k Probability}. This process repeats until the last multiplexer routes the CLASS_LABEL with the highest probability declaring it as the traffic class of the current packet.

VIII. EXPERIMENTAL RESULTS

In this section, the results obtained using the various experiments described in Sections VI and VII are discussed. This section discusses the software-based classifier performance followed by the FPGA implementation and results.

TABLE 7. UNIBS F-scores of the 10-fold cross-validation experiment.

Feature Subset	Naïve Bayes	Linear SVM	Polynomial SVM	KNN	Random Forest
All Features	0.7988	0.9794	0.9795	0.9864	0.9936
Ports Only	0.9145	0.9794	0.981	0.9878	0.9907
SWR	0.6281	0.9794	0.9772	0.9889	0.9939
RF	0.7704	0.9794	0.9751	0.9862	0.9933
All Features (No Ports)	0.6789	0.6523	0.8157	0.7568	0.9258
SWR (No Ports)	0.5068	0.6164	0.7316	0.7554	0.9103
RF (No Ports)	0.6736	0.6432	0.7863	0.7566	0.9282

A. SOFTWARE-BASED CLASSIFIER PERFORMANCE

Table 7 and Fig. 8 show the F-scores of the different classifiers in a 10-fold cross-validation experiment on the UNIBS dataset using the scikit-learn library in Python. The UNB F-scores tend to show similar trends, hence only the UNIBS results are shown for brevity. The experiments are repeated using the different feature subsets reported in Table 6. Note that, the feature subsets denoted with $*$ are the ones without port numbers. F-score results tend to highlight the difference in performance between random forest and all other classifiers. This is obvious in the port-less cases where random forest is always above 0.9 whereas other classifiers are almost always below 0.8. Besides, such a high F-score suggests that random forests are not overfitting to the training data. Another observation is that the features selected by the random forest feature selection algorithm yield a higher F-score than the "all features without port numbers" case using a random forest classifier. This is when feature selection boosts the classification performance after removing irrelevant features. Also, RF features tend to provide slightly better results when compared to the SWR features.

To further investigate the performance of the random forest classifier, the best performer, its behavior is examined in terms of false positive, false negative, precision, and recall.

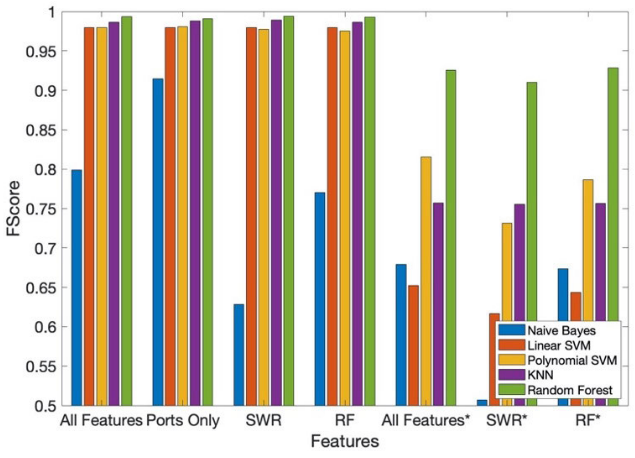

FIGURE 8. UNIBS f-score.

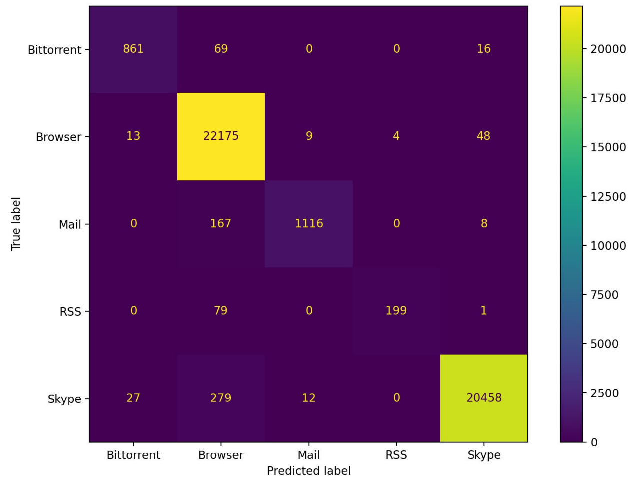

FIGURE 9. Random forest confusion matrix.

Precision is the percentage of packets that were labeled positive and are originally positive, while recall is the percentage of positive packets that were labeled positive. Fig. 9 shows the confusion matrix resulting from the 10-fold random forest cross-validation experiment using the UNIBS dataset. In this experiment, the features retained by the random forest feature selection algorithm without port numbers were used. Table 8 summarizes these key performance measures. The table shows that the random forest model has a very high probability of classifying most of the packets correctly. Nevertheless, the recall results indicate that the model is weaker at classifying Mail and RSS packets since many positive packets were mislabeled compared to other classes. This could be explained by the class imbalance problem of the UNIBS dataset resulting in slightly poorer performance on classes with a lower number of packets.

To ensure that the previously mentioned cross-validation results were not obtained through chance the Student's t-test for dependent samples is performed between the different folds to verify whether they have similar average values. If so, this would suggest that the obtained results could generalize

TABLE 8. Random forest performance measures.

Class	False Positive	False Negative	Precision (TP/TP+FP)	Recall (TP/TP+FN)
BitTorrent	40	85	0.956	0.91
Browser	594	74	0.974	0.997
Mail	21	175	0.982	0.864
RSS	4	80	0.98	0.713
Skype	73	318	0.996	0.985

well on future unseen network packets. The null hypothesis of the t-test states that the mean values of the different folds are identical. The selected significance level (alpha) is 0.05. The results prove that the p-values of all features across all the folds are considerably larger than the significance level. This implies that the null hypothesis that the mean values are identical cannot be rejected which further strengthens the claim that the results can generalize well to future unseen network packets.

The algorithms examined so far are traditional machine learning algorithms. To study a different dimension of the problem, the performance of more sophisticated systems such as Artificial Neural Networks (ANN) is investigated. Therefore, the UNIBS dataset was used to build a Multilayer Perceptron (MLP) artificial neural network using the features retained by random forest without port numbers. The ANN optimizes the log-loss function using a stochastic gradient-based optimizer where the activation function used is the logistic function and the number of hidden layers is one. The number of neurons in the hidden layer is the average of the number of input neurons and output neurons since it is common to use this average in the case of one hidden layer. Moreover, the dataset was normalized before feeding it into the ANN to avoid the oscillation of the learning process. The obtained classification accuracy is 92.9%, and the F-score is 0.644. These results fall behind the random forest model which further supports the claim that random forest classifiers are the best performers among the tested algorithms in the domain of network traffic classification.

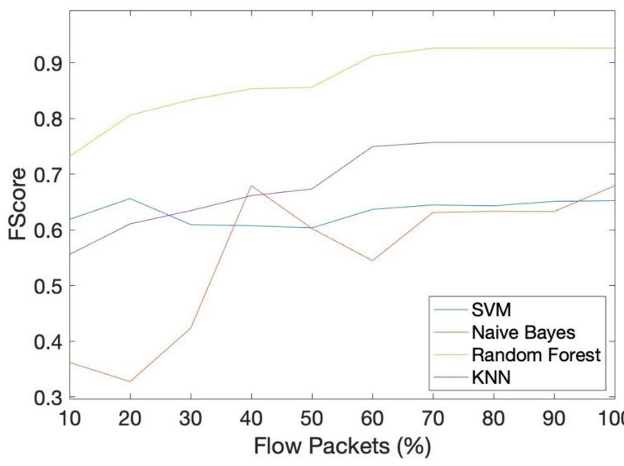

FIGURE 10. UNIBS all features (without ports) – f-score vs. percentage of considered packets within a flow.

Furthermore, Fig. 10 shows clearly that as the percentage of packets considered for flow-level feature extraction is increased, the accuracy of the classification increases. It is obvious that random forest starts with an F-score of almost 0.74 at 10% of flow packets, but it eventually reaches around 0.93 at 100% of packets. This plot also shows the wide gap of performance between random forest and its next competitor, KNN. It is also noticeable from this figure that the F-scores of the best performers, random forest and KNN, saturate at almost 60% of packets in a flow with KNN reaching an F-score of about 0.75 and random forest reaching an F-score of approximately 0.91. Therefore, it seems reasonable to choose 60% of the packets within a flow to be able to classify the flow with high accuracy and F-scores.

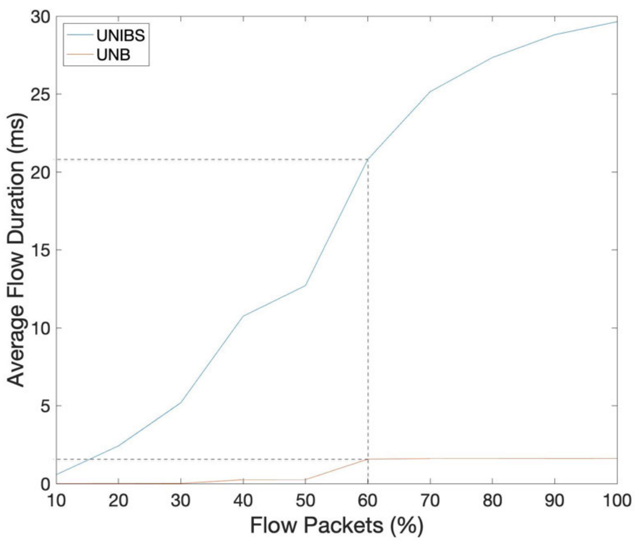

FIGURE 11. Average flow duration vs. percentage of considered packets within a flow.

To investigate the effect of packet percentage within a flow on the waiting time required to obtain the necessary packets prior to classifying a network flow, the average flow duration is plotted against the percentage of considered packets within a flow as shown in Fig. 11. It is found that the average required time to wait for 60% of the flow packets is around 21 ms using the UNIBS dataset. On the other hand, the reported results using the UNB dataset looks more like two flat straight lines with a very minor jump in between. Therefore, this endorses the previous arguments that the behavior of a real network cannot be accurately simulated using the UNB dataset since it was collected in a controlled environment.

B. HARDWARE-BASED CLASSIFIER PERFORMANCE

The scikit-learn library in Python was used to train the random forest classifier offline using both a majority-based model and a probability-based model. Unfortunately, the scikit-learn library does not produce complete trees, therefore, some levels in the trees might be missing nodes since leaf nodes appear at an earlier level of the tree. This is problematic with the hardware design since all trees need to have the same number of levels and the levels need to be complete to ease the hardware design. Therefore, a Python script was developed to train a random forest classifier while making sure that the produced trees are complete.

Fitting the entire random forest on the FPGA chip will be very difficult due to memory limitations and the limited number of Configurable Logic Blocks (CLBs) available on the chip. As a result, an experiment is conducted whereby the number of trees in the forest is varied from 1 to 50 trees (the optimal number of trees according to the previous experiments). Meanwhile, the maximum number of levels per tree that the FPGA chip can sustain is observed. After that, the classification F-score of the generated model is checked to find out the combination of trees and levels that yield the best classification performance when implemented in hardware.

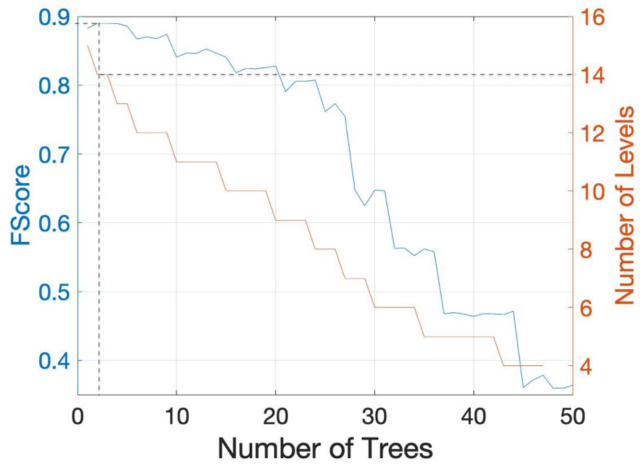

FIGURE 12. UNIBS f-score – effect of the number of trees and levels on f-score.

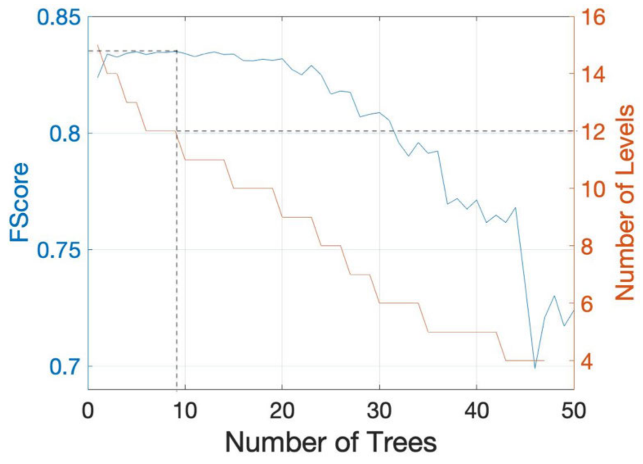

FIGURE 13. UNB f-score - effect of the number of trees and levels on f-score.

Fig. 12 and Fig. 13 show the F-score for the UNIBS and UNB datasets respectively, using the respective test sets. Upon inspecting the graphs, it is confirmed that when the number of levels in a tree reduces the classification performance drops. Therefore, the graphs show that fitting a greater number of trees on the FPGA chip means that the number of levels in each tree must be reduced which results in lower classification accuracy. Therefore, one of the objectives of this work is to find the optimal combination of the number of trees and the number of levels that yield the best classification performance. The graphs show that the F-score peaks at 2 trees each tree having 14 tree levels using the UNIBS dataset and 9 trees each with 12 tree levels using the UNB dataset.

TABLE 9. FPGA model vs. software optimal model for the UNIBS dataset.

Measure	Hardware		Software	
	Majority	Prob.	Majority	Prob.
Accuracy (%)	96.3	96.5	98.3	98.5
F-score	0.823	0.834	0.927	0.932
Kappa	0.926	0.929	0.968	0.969
AUC ROC	N/A	0.972	N/A	0.993

Table 9 shows the comparison in terms of classification accuracy, F-score, Cohen's kappa statistic, and Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) between the hardware-based models and the optimal software-based models obtained using Python for the UNIBS dataset. By default, ROC is defined for binary classification problems and is not suitable for multi-class problems like the problem in hand. Therefore, to expand it to multi-class problems the network traffic classification problem is binarized using the one-vs-one approach, and the weighted average mechanism is used to report a single ROC value to account for class imbalance. However, the AUC ROC cannot be reported as a single number for multi-class problems that use a majority-based classification. Hence N/A was inserted in the place of the AUC ROC for majority-based models. In all cases the probability-based (prob.) random forest models outperform the majority-based models. Even though going from software to hardware the number of trees reduced from 50 (optimal number of trees) to 20 and the number of levels reduced from 32 to 10, the changes in the performance measures were insignificant.

Table 10 shows the comparison between the hardware model (20-trees, 10-levels), the pruned software model (20-trees, 10-levels), and the fully-grown software model (20-trees, fully grown) using the UNIBS dataset. The hardware results are identical to the pruned software results due to the precision of the Python script in converting the trained software model to its hardware counterpart. Also, the comparison between the hardware performance and the fully-grown software performance reassures that not much performance is lost when using a pruned tree.

It is also essential to perform a timing analysis of the synthesized circuit to ensure that it meets all timing requirements. The maximum frequency at which the proposed random forest design can operate is found to be approximately 35 MHz. Hence, the period is calculated as 28.571 ns. As a result of using a highly pipelined architecture, one classification can

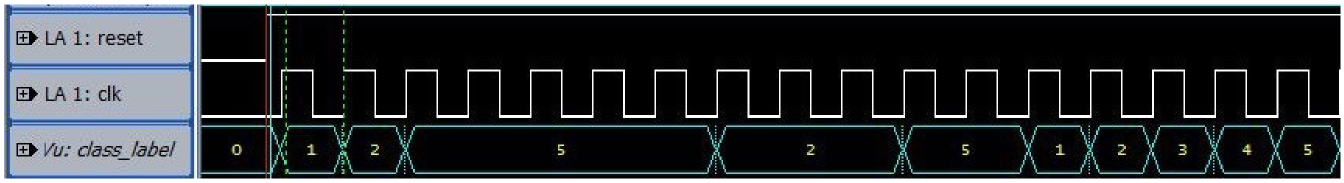

FIGURE 14. Logic analyzer's waveform.

TABLE 10. FPGA model vs. pruned software model vs. fully-grown software model for the UNIBS dataset.

Measure	Hardware		Pruned Software		Fully-Grown Software	
	Majority	Prob.	Majority	Prob.	Majority	Prob.
Accuracy (%)	96.3	96.5	96.3	96.5	98.3	98.4
F-score	0.823	0.834	0.823	0.834	0.924	0.933
Kappa	0.926	0.929	0.926	0.929	0.968	0.973
AUC ROC	N/A	0.972	N/A	0.972	N/A	0.984

be obtained every clock cycle once the pipeline is full. This means that one packet is classified every 28.571 ns. On the other hand, the average time taken to classify a packet in software was found to be 2.6469 μs and 1.3624 μs using the UNIBS and UNB datasets, respectively. The difference in classification time results from the different structures of the random forest trees built using the two datasets. These numbers reflect the enormous speedup obtained using the hardware acceleration of the random forest algorithm. The speedup can be defined as the increase in execution speed when using one system in place of another. Mathematically, the speedup can be calculated using (2):

$$\text{Speedup} = \frac{\text{Software} - \text{Based Classification Time}}{\text{Hardware} - \text{Based Classification Time}} \tag{2}$$

UNIBS Speedup = 2.6469 μ s / 28.571 ns = 92.64
UNB Speedup = 1.3624 μ s / 28.571 ns = 47.68

Therefore, compared to the software implementation of a random forest algorithm, the hardware accelerator can be up to 92 times faster when classifying a packet in the UNIBS dataset, and 47 times faster when classifying a packet in the UNB dataset. This is an encouraging result since the objective is to deploy such an accelerator at data centers that handle millions of network packets per second. To find out the average throughput achieved by the design, the average packet size of the UNIBS and UNB datasets was found to be 626 bytes per packet using Wireshark. Using a 35 MHz clock means that approximately 35 million packets can be classified per second. This results in a total throughput of 35M (packets classified per second) multiplied by 626 (average packet size) which translates to a throughput of 163.24 Gbps. Hence, the average throughput achieved by the proposed random forest design is 163.24 Gbps.

To further verify the operation of the random forest classifier, the Tektronix LA6401 logic analyzer is used to inspect the signals of the random forest module on the FPGA. This helps analyze, test, and debug the circuit in case of any misbehavior. To do so, the important signals of the random forest module are projected onto the GPIO pins of the DE2-115 board. Those signals include the master clock, the reset, and the class label signals. After that, the logic analyzer's probes are connected to the GPIO pins to record the behavior of those signals. Note that, the packets used are known to be of classes 1, 2, 3, 4, and 5, respectively.

Fig. 14 shows the waveform obtained using the logic analyzer where the clock signal runs at 35 MHz. After hitting the reset signal, the first packet is passed to the random forest classifier at the first negative edge of the clock. After 12 clock cycles, which is the time it takes the packet to pass through all pipeline stages in the design, the module starts producing the class label of the first packet followed by the successive classes. Notice that, the output during the first 12 clock cycles is treated as a "do not care." Table 11 shows a summary of the FPGA resource utilization after implementing the (20-trees, 10-levels) random forest model.

TABLE 11. FPGA resource utilization.

Resource	Total	Used	% Utilization
Configurable Logic Blocks (CLBs)	114480	79207	69%
Total Registers	N/A	1200	N/A
Total Pins	529	9	2%
Total Virtual Pins	N/A	0	0%
Total Memory Bits	3981312	1258260	32%
Embedded Multiplier 9-bit Elements	532	0	0%
Total PLLs	4	0	0%

Fig. 15 shows the final prototype of the random forest network traffic classifier implemented on the DE2-115 board. The seven-segment display is used to output the current packet's traffic class to the user. In this figure, it is observed that the current packet is of class 5 (Skype).

Table 12 shows a summary of the obtained accuracies and F-scores from the literature compared to the ones obtained in our experiments. Note that some of the reviewed work did not report the F-score results, which made the comparison slightly more difficult as two papers surpassed the obtained F-score, while one paper fell behind. On the other hand, the proposed design surpasses all the reviewed work in terms of classification accuracy.

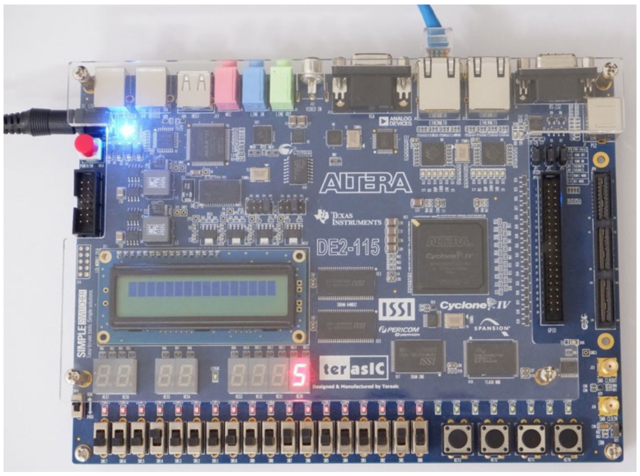

FIGURE 15. Class 5 on the DE2-115 board.

TABLE 12. Summary of accuracies and F-scores in the literature.

Design	Accuracy (%)	F-score
FPGA-Based RF	98.5	0.932
[4]	98	0.98
[11]	96.3	N/A
[12]	95.5	N/A
[17]	90.6	0.964
[18]	91.3	0.916
[19]	97.5	N/A

TABLE 13. Summary of throughputs in the literature.

Design	Throughput (Gbps)
FPGA-Based RF	163.24
[20]	8
[21]	28.6
[22]	40
[23]	80

The maximum throughput achieved due to implementing the random forest algorithm on an FPGA is also examined. Although this work, to the best of our knowledge, is the first attempt to accelerate a random forest-based network traffic classifier on an FPGA, nonetheless, the achieved throughput is compared to that of the other implementations including C4.5 decision tree and SVM based classifiers. Table 13 shows a summary of the obtained throughputs from the literature. The maximum throughput achieved by the proposed random forest accelerator is 163.24 Gbps. This is more than twice as fast as the maximum reported throughput in Table 13.

C. DISCUSSION OF RESULTS

Traffic classification is a non-linearly separable problem since it is very difficult to linearly differentiate between packets that travel across the network; therefore, a non-linear classifier was needed. This is the reason why naïve Bayes, which leads to a linear decision boundary in most cases, is the worst among all tested classifiers. The second worst performer is linear SVM, another linear classifier, which tries to determine yet another linear decision boundary that separates between the classes under investigation. Moving from linear SVM to just second-order polynomial SVM enhances the performance significantly which further endorses the previously mentioned claim. The same justification applies to KNN which attempts to find a complex non-linear decision boundary to distinguish the different classes. That is why the results of KNN are comparable to 2nd order polynomial SVM.

As the design moves away from linear classifiers and towards non-linear ones, the classification performance is improved significantly. Decision trees are known to be very suitable for non-linearly separable domains. Hence, many researchers proposed the use of decision tree-based classifiers like C4.5 [7]. Nevertheless, one of the main features of the random forest algorithm is its ability to generalize well and avoid overfitting to the training set when compared to single decision trees like C4.5. Moreover, random forests are less influenced by outliers than the previously mentioned algorithms. Additionally, random forests do not make any assumptions about the distribution of the extracted features, and hence they handle possible collinearity between the features. This in turn enables random forests to build more predictive models that are not impacted by such collinearity.

To the best of our knowledge, researchers have not investigated a random forest-based network classifier on FPGAs, hence this was a great opportunity to add to the literature body. Furthermore, the fact that the structure of random forest trees relies heavily on parallelism since each decision tree within the forest operates independently and in parallel with the other trees, was a very intriguing idea in conjunction with the ability of FPGAs to support highly parallel architectures. Therefore, FPGAs are the most suitable accelerators for random forests since they both have the idea of parallelism at their core structures.

On the other hand, ANN and deep learning models are very powerful, yet they are not suitable for every application. Deep learning is usually superior at handling more complex problems that contain lots of unstructured data such as image processing, natural language processing, speech recognition, and many others. However, well-defined problems like traffic classification which consist of structured data (features are organized in packet headers) do not need a complex feature engineering process. Hence, it is more likely that traditional machine learning algorithms will perform better and will certainly consume fewer resources and time compared to deep learning ones.

IX. CONCLUSION

In this article, several experiments were conducted on two publicly available datasets, UNIBS and UNB, whereby the traffic traces were run through several machine learning algorithms like naïve Bayes, linear SVM, polynomial SVM, KNN, random forest, and artificial neural networks.

The suitability of the extracted feature variables was validated using stepwise regression and random forest feature selection. Random forest resulted in the highest classification accuracy of 98.5% and an F-score of 0.932. It was proved that the most optimal percentage of packets within a flow that needs to be considered when extracting flow-level features is around 60% which required a waiting time of about 21 ms using the UNIBS dataset. Moreover, a random forest network traffic classifier hardware accelerator was designed on the DE2-115 FPGA board that consists of 20 trees each having 10 tree levels. Results show that the FPGA-based random forest traffic classifier achieves 96.5% accuracy and 0.834 F-score. Furthermore, the speedup obtained using hardware acceleration compared to software reaches 92.64 and 47.68 on the UNIBS and UNB datasets, respectively, while boosting the classification speed significantly resulting in average throughput of 163.24 Gbps. The achieved throughput enables the real-life deployment of the proposed design at data centers operating at 10s and 100s of Gbps.

ACKNOWLEDGMENT

This article represents the opinions of the authors and does not mean to represent the position or opinions of the American University of Sharjah.

REFERENCES

[1] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar, "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1988–2014, 2nd Quart., 2019.

[2] Y. R. Qu and V. K. Prasanna, "Compact hash tables for high-performance traffic classification on multi-core processors," in *Proc. IEEE 26th Int. Symp. Comput. Archit. High Perform. Comput.*, Oct. 2014, pp. 17–24.

[3] T. Soylu, O. Erdem, A. Carus, and E. S. Guner, "Simple CART based real-time traffic classification engine on FPGAs," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Cancun, Mexico, Dec. 2017, pp. 1–8, doi: 10.1109/RECONFIG.2017.8279820.

[4] G. Lu, R. Guo, Y. Zhou, and J. Du, "An accurate and extensible machine learning classifier for flow-level traffic classification," *China Commun.*, vol. 15, no. 6, pp. 125–138, Jun. 2018.

[5] B. Yamansavascilar, M. A. Guvensan, A. G. Yavuz, and M. E. Karsligil, "Application identification via network traffic classification," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Santa Clara, CA, USA, Jan. 2017, pp. 843–848.

[6] A. Este, F. Gringoli, and L. Salgarelli, "On-line SVM traffic classification," in *Proc. 7th Int. Wireless Commun. Mobile Comput. Conf.*, Istanbul, Turkey, Jul. 2011, pp. 1778–1783.

[7] D. Tong, Y. R. Qu, and V. K. Prasanna, "Accelerating decision tree based traffic classification on FPGA and multicore Platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3046–3059, Nov. 2017.

[8] T. Groléat, M. Arzel, and S. Vaton, "Stretching the edges of SVM traffic classification with FPGA acceleration," *IEEE Trans. Netw. Service Manage.*, vol. 11, no. 3, pp. 278–291, Sep. 2014.

[9] Z. Zhu, A. X. Liu, F. Zhang, and F. Chen, "FPGA resource pooling in cloud computing," *IEEE Trans. Cloud Comput.*, early access, Oct. 4, 2018, doi: 10.1109/TCC.2018.2874011.

[10] (Jul. 12, 2011). *UNIBS: Data Sharing*. UNIBS. Accessed: Feb. 14, 2019. [Online]. Available: http://netweb.ing.unibs.it/~ntw/tools/traces

[11] M. Dusi, F. Gringoli, and L. Salgarelli, "Quantifying the accuracy of the ground truth associated with Internet traffic traces," *Int. J. Comput. Telecommun. Netw.*, vol. 55, no. 5, pp. 1158–1167, Apr. 2011.

[12] F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, F. Risso, and K. C. Claffy, "GT: Picking up the truth from the ground for Internet traffic," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 5, pp. 8–13, Jan. 2009.

[13] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and VPN traffic using time-related features," in *Proc. 2nd Int. Conf. Inf. Syst. Secur. Privacy*, Rome, Italy, 2016, pp. 407–414.

[14] Y. Wang, Y. Xiang, J. Zhang, and S. Yu, "Internet traffic clustering with constraints," in *Proc. 8th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Limassol, Cyprus, Aug. 2012, pp. 619–624.

[15] T. Shanableh and K. Assaleh, "Feature modeling using polynomial classifiers and stepwise regression," *Neurocomputing*, vol. 73, nos. 10–12, pp. 1752–1759, Jun. 2010.

[16] X. Lin, R. D. S. Blanton, and D. E. Thomas, "Random forest architectures on fpga for multiple applications," in *Proc. Great Lakes Symp. VLSI*, Banff, AB, Canada, May 2017, pp. 415–418.

[17] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining Practical Machine Learning Tools and Techniques*, 4th ed. Cambridge, MA, USA: Morgan Kaufmann, 2017. Accessed: May 23, 2019. [Online]. Available: https://www.sciencedirect.com

[18] U. M. Fayyad and K. B. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *Proc. 13th Int. Joint Conf. Artif. Intell.*, Chambéry, France, 1993, pp. 1022–1027.

[19] L. Wang, L. Peng, M. Su, B. Yang, and X. Zhou, "On the impact of packet inter arrival time for early stage traffic identification," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. (CPSCom) IEEE Smart Data (SmartData)*, Chengdu, China, Dec. 2016, pp. 510–515.

[20] A. Monemi, R. Zarei, and M. N. Marsono, "Online NetFPGA decision tree statistical traffic classifier," *Comput. Commun.*, vol. 36, no. 12, pp. 1329–1340, Jul. 2013.

[21] T. Groleat, M. Arzel, and S. Vaton, "Hardware acceleration of SVM-based traffic classification on FPGA," in *Proc. 8th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Limassol, Cyprus, Aug. 2012, pp. 443–449.

[22] S. Valenti, D. Rossi, A. Dainotti, A. Pescapè, A. Finamore, and M. Mellia, "Reviewing traffic classification," *Data Traffic Monitoring and Analysis*, vol. 7754, no. 1. Berlin, Germany: Springer, 2013, pp. 123–147, doi: 10.1007/978-3-642-36784-7_6.

[23] W. Jiang and V. K. Prasanna, "Large-scale wire-speed packet classification on FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, California, CA, USA, 2009, pp. 219–228.

MOHAMMED ELNAWAWY was born in Giza, Egypt, in 1995. He received the B.Sc. *(summa cum laude)* and M.Sc. degrees in computer engineering from the American University of Sharjah, United Arab Emirates, in 2017 and 2019, respectively. He worked as a Graduate Teaching Assistant and a Research Assistant with the American University of Sharjah, where he joined the Department of Computer Science and Engineering, as a Laboratory Instructor, in 2020. His research interests include machine learning, field-programmable gate arrays, and embedded systems. He is a member of the Upsilon Pi Epsilon, honor society. He won several awards, including the American University of Sharjah Graduate Student Research, scholarly, the Creative Work Excellence Award, and the Sharjah Islamic Bank Research Award.

ASSIM SAGAHYROON (Senior Member, IEEE) received the M.Sc. degree in electrical engineering from Northwestern University, Evanston, IL, USA, and the Ph.D. degree from The University of Arizona, Tucson, AZ, USA. From 1993 to 1999, he was a member of the Department of Computer Science and Engineering, Northern Arizona University. In 1999, he joined the Department of Mathematics and Computer Science, California State University. In 2003, he joined the Department of Computer Science and Engineering, American University of Sharjah, where he served as the Department Head for seven years. He was an Invited Technical Reviewer for some of the National Science Foundation programs and served on the technical program committees of many conferences. He has many publications in international conferences and journals. His research interests include innovative applications of emerging technology in the medical field, power consumption of portable electronics, and FPGAs-based design.

TAMER SHANABLEH (Senior Member, IEEE) was born in U.K., in 1972. He received the B.Sc. degree in computer science from Mutah University, Jordan, in 1994, and the M.Sc. degree in software engineering and the Ph.D. degree in electronic systems engineering from the University of Essex, in 1998 and 2002, respectively. From 1998 to 2001, he was a Senior Research Officer with the University of Essex, during which he collaborated with BTexact on inventing video transcoders. He joined the U.K. Research Laboratory, Motorola, in 2001. During his affiliation at Motorola, he contributed to establish a new profile within the ISO/IEC MPEG-4 known as the Error Resilient Simple Scalable Profile. He joined the American University of Sharjah, in 2002, where he is currently a Professor of Computer Science. During his summer breaks, he worked as a Visiting Professor at Motorola Laboratories for five different years. He spent the spring semester of 2012 as a Visiting Academic with the Multimedia and Computer Vision Laboratory, Queen Mary University of London, U.K. He is also a Professional Engineer. He has authored around 80 publications and has six patents. His research interests include digital video processing and pattern recognition.