Pipelined Decision Trees for Online Traffic Classification on FPGAs

Oğuzhan Erdem[1], Tuncay Soylu[2] and Aydin Carus[3]

[1]*Electrical and Electronics Engineering, Trakya University, Edirne, TURKEY 22030*
[2]*Occupational Health and Safety, University of Health Sciences, Istanbul, TURKEY 34668*
[3]*Computer Engineering, Trakya University, Edirne, TURKEY 22030*
Email: ogerdem@trakya.edu.tr

Decision tree-based Machine Learning (ML) algorithms are one of the preferred solutions for real-time internet traffic classification in terms of their easy implementation on hardware. However, the rapid increase in today's newly developed applications and the resulting diversity in internet traffic greatly increases the size of decision trees. Therefore, the tree-based hardware classifiers cannot keep up with this growth in terms of resource usage and classification speed. To alleviate the problem, we propose to group application classes by certain rules and create an individual small decision tree per each group. In this article, a pipelined organization of multiple decision tree data structures, called Pipelined Decision Trees (PDT), is proposed as a scalable solution to tree-based traffic classification. We also propose two distinct algorithms, namely Confusion Matrix-based Class Aggregation (CMCA) and Leaf Count-based Class Aggregation (LCCA) algorithms, to set group creation rules that allows traffic classification on pipelined smaller decision trees in a hierarchical order. We further designed an hardware engine on Field Programmable Gate Arrays (FPGAs), that can search those pipelined trees within a single clock cycle by transforming them into bit vectors and implementing multiple range comparisons in parallel. Our architecture with 12 classes can run in 928.88 giga bit per second (Gbps) and achieve 96.04% accuracy.

Keywords: Traffic Classification; Machine Learning; Decision-tree; Data structure; FPGA; Flow association

1. INTRODUCTION

Network traffic classification is the key function of network security and management processes such as ensuring quality of service (QoS), resource usage projection, identifying security threats in intrusion detection frameworks and traffic shaping/billing needed. In traffic classification, an incoming network traffic is associated with the application classes that generate the traffic so that appropriate network service is provided according to predefined agreements or rules. If this process is performed while traffic is flowing, it is called real-time traffic classification [1].

The techniques developed for internet traffic classification can be examined in four groups; (*i*) Port number based, (*ii*) Deep packet inspection (DPI) based, (*iii*) Heuristic based and (*iv*) Machine learning (ML) based. In *port number based* approaches, only TCP or UDP port numbers of internet packets are processed to determine the application class of the flowing traffic. However, it is a fact that some new applications may not have their dedicated port numbers enrolled in Internet Assigned Numbers Authority (IANA), while others specifically prefer to use unpredictable dynamic port numbers to hide themselves [2]. As a result, the classification accuracy of port number based approaches is adversely affected with the increase in the usage of unsteady port numbers. As the techniques utilizing port numbers often cause erroneous predictions, solutions have begun to be developed in the field of Intrusion Detection Systems (IDS) [3]. In *DPI based* approaches, signature analysis is applied to recognize specific signatures of applications [4, 5]. However this method does not yield the expected results under encrypted traffic conditions [6, 7, 8]. Furthermore, the interest in DPI based approaches has waned due to government-enforced privacy regulations that restrict third parties from legally examining package contents [9]. *Heuristic based* techniques classify internet traffic according to specific traffic patterns. However, these approaches have large memory requirements for storing traffic patterns and their classification accuracy is limited compared to existing approaches [10]. In order to cope with all those aforementioned problems, *Machine Learning (ML) based* techniques that can classify internet traffic using the statistical properties of traffic flows without the need for content information has emerged as

a critical solution in this area [11, 12, 13, 14, 15]. In ML based approaches, a traffic classification model is designed using a pre-classified or labeled set of flows. The new incoming unknown streams are then classified according to this created model. ML based classification approaches have attracted great interest in recent years, especially because they can work in encrypted traffic conditions and Internet of Things (IoT) environments while providing cybersecurity solutions with high performance outcomes [16, 17, 18, 19, 20, 21].

In order to perform online traffic classification within ML based algorithms, decision trees (DTs) are the mostly preferred choice due to their layered structure and easy mapping onto pipelined hardware [10, 11, 16, 22]. C4.5 in particular, is one of the most frequently studied algorithm among decision trees due to its high classification accuracy and quick adaptability to hardware [11, 23, 24, 25]. In DT based solutions, the depth of a tree is closely related to the number of traffic application classes. As the number of classes increases, more branching to distinguish those classes is needed, and resultantly the tree depth increases further. This situation negatively affects the search delay as well as the throughput performance. As a solution, data structure transformation techniques that make the tree depth independent of the number of classes while without changing the classification philosophy are utilized [16]. In this way, one can accomplish real-time traffic classification without performance degradation due to the likely growth of internet applications in the near future.

In our previous study, we deeply focused on this problem and proposed a novel data structure called Bit Vector-Coded Simple Cart (BC-SC) [16], which is an alternative representation of a Simple CART (SC) [26] decision tree. In BC-SC, a single large decision tree is transformed into many small range trees with the help of bit vectors, and the depths of these new trees are independent of the number of application classes. Furthermore, we proposed a novel Discrete Parallel Range Comparators (DPRC) based hardware architecture to support BC-SC data structure. Typically, the pipeline model is used for hardware mapping of tree structures and the search process is done separately at each tree level in the form of a pipeline. On the other hand, DPRC-based BC-SC can search all the tree nodes simultaneously within a single step regardless of their levels, once the tree nodes are disjoint. Thus BC-SC can support large number of application classes while achieving real time classification within only a single clock cycle. However, we observed that the most important factor determining the real throughput is the size of the bitmaps used in the transformation whereas long bit vectors cause large hardware critical path delay and thus increases the clock cycle period. This paper hereby concentrates on this observation and addresses the problem with a novel traffic classification engine, named as Pipelined Decision Trees (PDT) architecture. PDT employs multiple smaller trees having few number of leaf nodes to set bounds to the growth of bitmap sizes. The proposed scheme can support any kind of decision tree structure while providing substantial throughput improvement and reduced latency regardless of the number of application classes. The following major contributions are done in this article:

- We have developed a novel data structure, named as Pipelined Decision Trees (PDT), as an alternative multi-level decision tree representation. The new structure prevents the instability of classical decision trees in node distributions and excessive growth in tree sizes due to the potential increase in the number of application classes in the short run. Thus, our proposal solves the most fundamental problems of implementing decision trees on hardware and facilitates real-time classification (Section 3.2).
- To construct PDT structure, we propose to aggregate the application classes in groups according to certain rules and create an individual tree structure for each group. We propose two alternative novel clustering algorithms, namely Confusion Matrix-based Class Aggregation (CMCA) and Leaf Count-based Class Aggregation (LCCA). A confusion matrix information which summarizes the prediction results of a classification problem is used to create groups of classes in CMCA algorithm (Section 4.1). The groups are organized based on the numbers of leaf nodes belonging to each class in the original decision tree in LCCA algorithm (Section 4.2).
- We further proposed 3 different alternative version of CMCA algorithm; (i) row based (r-CMCA) (Section 4.1.1), (ii) row/column sum based (s-CMCA) (Section 4.1.2) and (iii) percentage based (p-CMCA) (Section 4.1.3).
- We designed scalable, high throughput and low latency hardware architecture on Field Programmable Gate Array (FPGA) platform to support PDT data structure (Section 6). The proposed architecture with 12 classes reaches 928.88 Gbps (2902.76 MCPS) throughput with the minimum packet size of 40 Bytes while achieving an accuracy of 96.04% (Section 7).

We organized the remaining of the paper as follows: Section 2 presents the background information and related studies about traffic classification. In Section 3, PDT data structure is introduced. Class aggregation algorithms are explained in detail in Section 4. The hardware implementation issues of PDT is presented in Section 6. Section 7 demonstrates the performance evaluation of our proposed algorithms and FPGA-based designs. Finally, Section 8 concludes the paper.

2. BACKGROUND

2.1. Traffic classification overview

Internet traffic classification is the process of separating Internet Protocol (IP) traffic into predetermined classes using classical *packet-level* (source/destination port numbers, protocol etc.) or *flow-level* (average packet size, minimum/maximum packet size, inter-arrival times etc.) attributes or features in a machine learning terminology. In traffic classification, a *flow* is defined as a series of packets with the same 5-tuple header fields (Source IP address (SA), Destination IP address (DA), Source Port Number (SP), Destination Port Number (DP) and Protocol). Flow-level features are generally derived from the first R packets of the flow and the R value is chosen as a number that best represents the entire flow. Both packet-level and flow-level attributes are usually used together in different combinations to achieve high accuracy traffic classification even in encrypted traffic conditions.

2.2. Machine Learning (ML) based traffic classification

The use of machine learning techniques in network traffic control started with the NetMan program developed in the 1990s for the call completion maximization in a circuit-switched telecommunications network [27]. The study in [28], that surveys the use of artificial intelligence techniques in Intrusion Detection (ID) systems and presents an example of network connection classification, forms the basis of many subsequent papers experimenting ML algorithms in traffic classification.

Kim et al. evaluate the performance of CoralReef (ports-based), BLINC (host-behavior-based), and seven popular ML algorithms (flow-features-based) such as Naive Bayes (NB), NB classifier using kernel estimation (NBKE), Bayesian Network (BayesNet), C4.5 decision tree, k-Nearest Neighbors (k-NN), Artificial Neural Networks (ANN), Support Vector Machines (SVM) with anonymized payload traces and conclude that SVM outperformed all other methods on every trace over 98% accuracy [23]. In [24], the robustness of the models created by AdaBoost, SVM, NB, RIPPER and C4.5 algorithms were investigated for distinguishing SSH traffic from non-SSH traffic while utilizing from flow based features from four different network data sources. The results show that C4.5 based classifier performs best with 97% detection rate. SVM based traffic classifier proposed by [29] is experimented with three sets of traffic traces and over 90% accuracy is observed in almost all cases. Lim et al. tested the discriminative power of flow features and the effect of discretization using the NB, k-NN, SVM and C4.5 algorithms [30]. The results indicate that k-NN and C4.5 significantly outperform other algorithms with every sort of flow features used and the entropy-based discretization substantially improves the classification accuracies of all algorithms. C4.5 decision-tree is transformed into multiple compact tables in [31]. The proposed model employing efficient hashing techniques to minimize the processing latency achieves 98.15% classification accuracy with a typical C4.5 tree with 92 leaf nodes and seven flow-level features.

Caicedo-Munoz et al. proposed a process for VPN and Non-VPN traffic classification and achieved accuracy ratios over 94.42% using various ML-based classifiers with time-related features [32]. A model based on NB was proposed to classify three applications, that comprises two video services (Netflix and YouTube) and one file download. The results proves that the proposed algorithm is much faster than Gaussian NB in training and classification while providing average accuracy of 98.88% [33]. A hybrid model consisting of K-Means and RF classifiers is proposed to classify the set of five user activities and achieved an average accuracy of 97.37% [34]. Dong proposed enhanced SVM algorithm, namely cost-sensitive SVM (CMSVM) that targets to resolve data instability problems and decrease computational cost [35]. The algorithm reaches 94.2% and 94.5% maximum accuracy for the datasets MOORE_SET (10 applications) and NOC_SET (9 applications) respectively. The performances of single and ensemble models including k-NN, Gradient Boosting (GB), RF, Logistic Regression (LR), NB, MLP, AdaBoost and Bagging Decision Tree (BDT) were compared under P2P network traffic condition [36]. The results proved that ensemble algorithms of RF and BG outperforms the single algorithms in both VPN and non-VPN networks. An ensemble model CARD-B proposed by [37] comprises Capsule Neural Networks, Artificial Neural Networks (ANN), RF and DT classifiers with boosting techniques. The proposed design shows an overall accuracy of 96% with seven classes. Caicedo-Munoz et al. [38] compared the performance of DT, RF and 1-D Convolutional Neural Network (CNN) in classifying Android malware traffic and reported that RF shows the best performance with 97% recall and 86% F-measure. LR, SVM and ANN models were implemented in Software-Defined Networking (SDN) environment to classify seven traffic applications and the experimental results show that ANN model reaches the best accuracy of 89% [39].

2.3. FPGA based traffic classification

Luo et al. proposed a method for hardware implementation of C4.5 tree on FPGAs that substantially reduce the worst-case number of memory accesses [40]. An FPGA-based architecture to accelerate k-NN algorithm is proposed in [41]. The design sustains 80 Gbps throughput with accuracy ratio over 99% to classify multimedia applications VoIP, Instant Messaging (IM) and IPTV. Groleat et al. designed a hardware acceler-

ated SVM classifier on FPGAs that operates in 10 Gbps speed rate [42]. Monemi et al. proposed a NetFPGA-based hardware architecture for DT classifier. The design with several optimizations runs in maximum frequency of 68 MHz [43].

Tong et al. implemented C4.5 tree with two alternative FPGA-based design where the one utilizes on-chip distributed RAM and other stores the classifier in block RAM [10]. The high throughput architecture sustains a throughput of 550 Gbps with eight application classes. The C4.5 decision-tree is transformed into multiple hash tables and implemented on FPGAs by Gandhi et al [22]. The designed classification engine achieves a throughput of 1654 million classifications per second (MCPS) with a tree consisting of 128 leaves. Tristan et al. proposed a hardware accelerator for SVM based traffic classification and achieved 473 Gbps rates with four class of traffic trace [44]. In [11], the C4.5 decision-tree is represented by a compact rule set table and mapped onto an FPGA-based 2-dimensional pipelined architecture. The post-place-and-route results show that the designed engine sustains a 645 MCPS throughput with eight applications. The two distinct algorithms to accelerate C4.5 tree based classifier are developed in [45]. The first one optimizes the original classifier whereas the second employs divide and conquer approach. Both algorithms are implemented on FPGAs and 10000+ and 8000+ MCPS throughput values are observed respectively. Elnawawy et al. proposed a pipelined architecture to implement random forest algorithm on FPGAs [13]. The results shows that an average throughput of 163.24 Gbps and accuracy of 98.5% are achieved using both packet and flow-level features with five classes.

Siracusano et al. proposed to use binary neural networks (BNNs) in IoT traffic classification and the designed model on NetFPGA that succeeds 40 Gbps in classifying the network traffic into ten classes while utilizing 17 flow-level features [46]. Soylu et al. are the first to adapt the Simple CART (SC) decision tree algorithm for FPGA-based internet traffic classification [1]. The proposed pipelined architecture accomplishes 557 Gbps with the accuracy of 96.8%. The authors also proposed a novel Simple CART forest data structure consisting of multiple parallel SC trees for real-time traffic classification and mapped onto the FPGA platform [47]. The designed architecture shows the throughput of 854 Gbps with 96.67% accuracy ratio. Bit Vector-Coded Simple Cart (BC-SC) as an alternative representation of a SC decision tree is introduced in [16]. SC tree is converted into multiple range trees by utilizing bit vectors. BC-SC is implemented both in pipelined model and with discrete parallel range comparator units on state-of-the-art FPGAs. The designed classification engines sustain 665 and 914 Gbps throughput respectively and reach 96.81% accuracy level with eight application classes.

3. PIPELINED DECISION TREE (PDT) BASED TRAFFIC CLASSIFICATION

3.1. Motivation

Due to its layered structure, the decision tree is the most suitable machine learning algorithm for pipelined hardware implementation [10, 11, 22]. However, we highlight two major problems with decision trees; (i) the unbalance of the node distribution and (ii) the growth of the tree sizes with the increasing number of application classes. Both of these problems cause significant difficulties in hardware implementation of trees. The decision trees are generally implemented in hardware as a pipeline, where each level of a tree corresponds to a single pipeline stage. Thus, the depth of a tree determines the length of the pipeline and implicitly the search delay. In both of aforementioned problems, it causes an increase in the depth of the tree, thus increasing the classification delay which strictly prevents real-time classification [23, 24].

To solve these problems, we proposed to transform a decision tree into multiple range trees with the help of bit vectors in our previous study [16]. As a result, while the transformed scheme performs the same function as the original decision tree, with the accuracy value remaining constant, the range trees have well balanced structure and their depths are independent of the number of classes.

3.1.1. Bit vector-coded decision tree (BC-DT)

Figure 1a presents a sample C4.5 decision tree with six classes ("Services", "P2P", "WWW", "Attack", "VOIP" and "Chat") and three features (F1, F2 and F3). As can be seen from the shape, the tree is unbalanced in terms of the density of the nodes in left and right branches. The depth of the left and right branches relative to the root node are 1 and 10 respectively. As the number of classes increases, this imbalance may worsen and depth may increase even more. Figure 1c shows the corresponding balanced range tree representation with the help of bit vector transformation tables given in Figure 1b. Bitmaps are created using unique feature boundary values that are extracted from the main decision tree. Let's take the bitmap tables of F3 in Figure 1b as an example. The specific ranges of the F3 attribute are (0.5 - 1.5) and (13 - 41) as can be seen in Figure 1a. The unique boundary values can be sorted on a number line as ($-\infty$, 0.5, 1.5, 13, 41 and $+\infty$). Then, all application class names in the leaves of the original decision tree are listed in a table beginning from the left ("Services", "P2P", "WWW" etc.). A bitmap for any range in columns is then constructed in such a way that a particular bit in a bit vector is set to "1" if its corresponding class falls in that range (or it is uncertain whether it falls within the specified range), and "0" otherwise. Let's consider the bitmap for the first range of ($-\infty$, 0.5). The bit vector takes a value

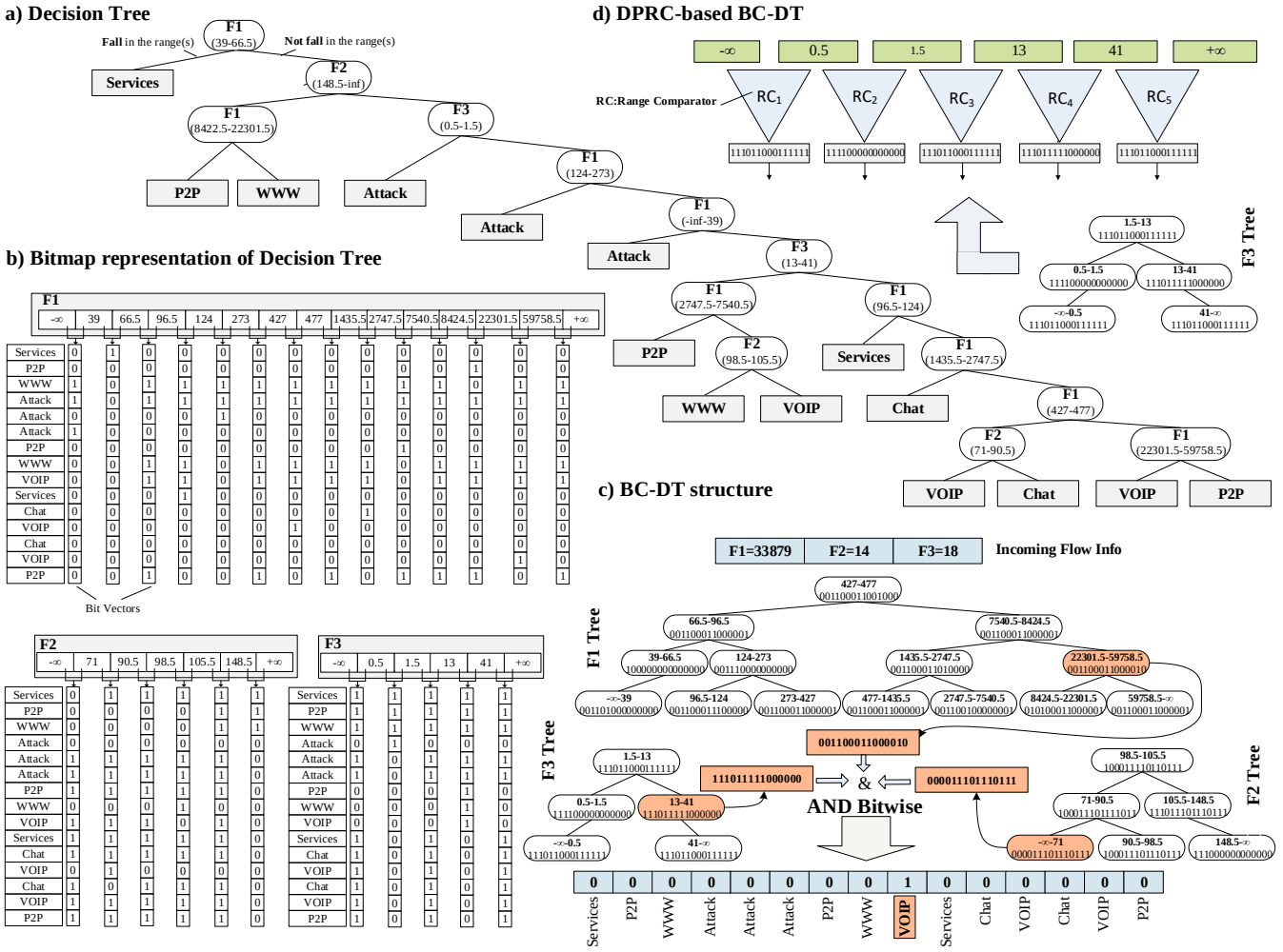

FIGURE 1. a) Simple Decision Tree (DT) b) Bitmap representation of DT c) Bit vector coded-DT (BC-DT) structure d) DPRC-based implementation of BC-DT

of 1 for "Services", "P2P" and "WWW" because it is unclear whether these classes fall within this range. However, since it is certain that the next "Attack" class does not fall within this range (it falls within the range of (0.5, 1.5)), the corresponding bitmap value is 0. Similarly, when this scan is performed for all the classes, a bit vector value of "111011000111111" is obtained. After the bitmaps are created for all features, they are stored in the binary range tree data structure, as shown in Figure 1c. A single range tree is created for each feature (separate F1, F2 and F3 trees for the corresponding features F1, F2 and F3). Due to the lack of space in the figure, we could not line up the trees side by side, but these trees are parallel to each other. Furthermore, as seen in the Figure 1c, all these trees are balanced and the depth of the largest tree is only 3 (whereas the main decision tree depth is 10). These range trees are constructed using the range values on the number line. First, the middle range interval is chosen as the root (the interval (1.5 - 13) for F3 tree as an example), and then the right and left subtrees are created iteratively. The range values of all nodes to the left of any node in the range tree are less than or equal to the lower bound value of this node. Similarly, the range limit values on all nodes to its right are greater than or equal to the upper range value on this node. Each node in a tree stores lower/upper range boundary values, left/right child pointers and the corresponding bitmap. Figure 1c also demonstrates sample search for incoming flow information F1=33879, F2=14, F3=18. Each attribute value is searched independently in its corresponding tree and an individual bitmap output (colored in orange in the figure) is obtained. Finally, the sample flow class is assigned (VOIP in our example) by combining the search results from three separate range trees with bitwise AND operation.

3.1.2. Hardware implementation of BC-DT

By taking advantage of the fact that the nodes of a range tree are disjoint, we can implement a tree in the form of a fully parallel search architecture with the help of range comparator (RC) units instead of the

pipeline structure in hardware (Discrete Parallel Range Comparators (DPRC)-based architecture in [16]). In this way, the latency depending on the number of stages in the usual pipeline search is reduced to a one-step range comparison delay with the feasibility of parallel simultaneous search of all distinct tree nodes. Figure 1d represents DPRC implementation of F3 Tree only where each node in a tree corresponds to a RC unit in the architecture. In this structure, since the intervals are discrete from each other, there is no need to store pointers as in tree nodes, that provides a memory advantage. Furthermore, the most critical benefit is that the nodes searched consecutively in the tree are searched simultaneously in parallel in DPRC. The incoming feature value is given as input to all comparator units, but only matches one of them and outputs the registered bitmap value. Normally, multiple sequential comparisons are performed when searching on the tree, a single step comparison is made by parallel search units. However, the time elapsed in a single-step range search, i.e. *clock period*, that determines the search throughput is also a major metric in hardware implementations besides the latency criterion. A clock period here is defined as the time it takes to perform one-step range comparison plus the time needed for merging the individual search results where there are as many intermediate result as the number of features. Although the range comparison process needs a constant duration, the time it takes to combine the search results increases due to the length of the bit vectors stored in the nodes in hardware. Note that, the bit vectors are created from leaves of the original decision tree and therefore their size is equal to the number of leaves in that tree (the bitmap length is 15 for the sample tree in Figure 1a). From this observation, we conclude that while the number of leaves in the original tree increases due the growth of the number of application classes, the throughput of the DPRC-based hardware architecture gets worse relatively.

3.2. Pipelined Decision Trees (PDT)

In this paper, we propose a new scalable model to considerably control the classification throughput besides the delay while eliminating the negative impact of likely increase in the number of traffic application classes in the future. Our approach is based on the fact that, instead of creating a single decision tree and representing it with larger bit vectors, we propose to create multiple small decision trees at the beginning and thus control the size of requiring bit vectors to adapt to the growth of internet applications in terms of the throughput. The way to build a small decision tree is simply to reduce the number of classes that can be achieved by grouping or merging multiple classes. In this paper, we propose to distribute application classes into separated groups and construct smaller decision trees with less number of classes. The process of reducing the number of classes by grouping is called *clustering* or *aggregating* interchangeably in the rest of the article. We also named the algorithms that perform clustering operations as *class aggregation algorithms* (CAA).

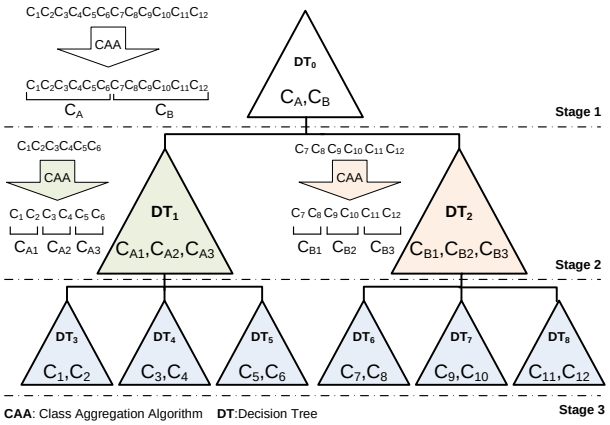

FIGURE 2. Pipelined Decision Trees (PDT) structure

In accordance with the output of clustering algorithms, we construct sequentially arranged multi-tree data structure, namely *Pipelined Decision Trees* (PDT), which is a layered structure and each layer contains one or more small decision trees as demonstrated in Figure 2. PDT consists of n+1 stages and each stage contains one or more decision trees where the parameter n represents the number of times the clustering algorithm is executed. The number of decision trees in any Stage k is determined by the number of groups created as the output of aggregation algorithm executed in Stage k-1.

Figure 2 represents 3-stage (n=2) PDT structure comprising the number of 1, 2 and 6 individual decision trees organized in pipelined fashion in Stage 1, 2 and 3 respectively. Let's consider a 12-class classification problem where class symbol C_i ($1 \leq i \leq 12$) stands for each class. When we apply the class aggregation algorithm for the first time in Stage 1, lets assume the algorithm combines the classes into two new classes as C_A and C_B, here C_A=$\{C_1,C_2,C_3,C_4,C_5,C_6\}$ and C_B=$\{C_7,C_8,C_9,C_{10},C_{11},C_{12}\}$. In this case, the decision tree in Stage 1 is constructed by only two classes C_A and C_B as shown in Figure 2. Once again we apply the aggregation algorithm for each group iteratively, let's assume that the groups C_{A1}, C_{A2} and C_{A3} are formed for the C_A, and C_{B1}, C_{B2} and C_{B3} are created for the C_B (where C_{A1}=$\{C_1,C_2\}$, C_{A2}=$\{C_3,C_4\}$, C_{A3}=$\{C_5,C_6\}$, C_{B1}=$\{C_7,C_8\}$, C_{B2}=$\{C_9,C_{10}\}$, C_{B3}=$\{C_{11},C_{12}\}$). In this case, each of the decision tree in Stage 2 comprises only three classes. Finally, since each C_{Ai} (C_{Bi}) has only two classes, the number of six decision trees each with 2 classes will take place in the last stage and the algorithm does not need to be executed again. As a result, instead of a single DT with large number of

leaves, a new structure with nine trees, each with a much lower number of leaves, is constructed.

In PDT structure, the search progresses by pipeline fashion starting from the root decision tree at the first stage to the specific tree at the last stage. According to the output of each stage search, which tree to traverse in the next stage is determined. For instance if the search result of Stage 1 in our example is found as class C_A in the root DT, then the tree on the left classifying C_{A1}, C_{A2} and C_{A3} is visited in Stage 2. The final classification result is obtained by the last searched DT. Each trees is searched by comparing the related feature input value with the values recorded in the nodes and the path is determined as the right or left accordingly. The search ends once a leaf node is reached.

4. CLASS AGGREGATION ALGORITHMS

We propose two new alternative clustering algorithms, namely *confusion matrix-based class aggregation* (CMCA) and *leaf count-based class aggregation* (LCCA) to group application classes based on predefined rules. In CMCA, a confusion matrix that summarizes the estimation results of classification problem is used to define the grouping strategy. We further proposed the three versions of CMCA algorithm; (i) *row based* (r-CMCA), (ii) *row/column sum based* (s-CMCA) and (iii) *percentage based* (p-CMCA). In LCCA, groups are formed according to the number of leaf nodes belonging to each class in the original decision tree. Note that, the choice of the best algorithm to use, the number of execution repetitions (n) that defines the number of stages and the determination of the number of groups in each stage are the optimization metrics to be decided for maximizing the classification accuracy particularly depending on the data.

4.1. Confusion matrix-based class aggregation algorithm (CMCA)

A confusion matrix presents the summary of the prediction results pertaining to a classification problem [48]. A two-dimensional confusion matrix displays the test set results in a row and column for each class. Figure 3 demonstrates a confusion matrix for four-class classification problem. Each element P_{ij} represents the number of test instances where the actual class C_i and the predicted class C_j are given in the row and column respectively. The diagonal elements P_{ii} in the matrix show the number of correctly predicted samples. T_{Rowi} shows the total number samples belonging to class C_i in a test set and T_{Coli} represents the total number of test examples predicted as C_i by the classifier. The sum of all P_{ij} elements is equal to the number of samples in the test set. The large numbers on the main diagonal and small numbers (ideally zero) corresponding to off-diagonal elements indicate good classification results. We have developed three alternatives of confusion matrix-based class aggregation algorithm (CMCA)

Actual Class \ Predicted Class	C_1	C_2	C_3	C_4	*Total*
C_1	P_{11}	P_{12}	P_{13}	P_{14}	T_{Row1}
C_2	P_{21}	P_{22}	P_{23}	P_{24}	T_{Row2}
C_3	P_{31}	P_{32}	P_{33}	P_{34}	T_{Row3}
C_4	P_{41}	P_{42}	P_{43}	P_{44}	T_{Row4}
Total	T_{Col1}	T_{Col2}	T_{Col3}	T_{Col4}	

FIGURE 3. Confusion matrix for four-class classification problem

that use the matrix information in different ways to form the groups of classes as described in the following sub-sections.

4.1.1. Row based (r-CMCA)

In row based (r-CMCA) algorithm, classes are initially ranked by looking at the P_{ij} values (i≠j) in the confusion matrix while the most frequently confused classes (having the largest P_{ij} values) are at the top of the list, and this order is taken into account when creating the groups. Initially, the maximum P_{ij} value is found by comparing the all P_{ij} values (i≠j) in the corresponding row and column of class C_i of the confusion matrix and this value is assigned to class C_i as the confusion strength value (CSV (C_i)). All the classes are then sorted from largest to smallest according to their CSV values. Finally all the classes are distributed into groups one by one while considering this sorted array. Note that, each group must have at least two classes and the number of groups is given to the algorithm as an input parameter. Algorithm 1 gives the pseudo code of proposed r-CMCA algorithm where the lines 1 to 22 sort classes based on their calculated CSV values and lines 23 to 33 distribute those classes in the sorted list into the groups.

Figure 4a shows the output confusion matrix of C4.5 decision tree given in Figure 1a which is created by training a real traffic data set (50 samples from each class and 300 samples in total) with 6 classes (Attack, Chat, P2P, Services, Voip and www). Figure 4b demonstrates the sorted list of classes with r-CMCA algorithm based on their CSV values. Figure 4c and d show the two alternative class distribution for the number of 2 and 3 groups respectively.

4.1.2. Row/column sum based (s-CMCA)

In row/column sum based (s-CMCA) algorithm, confusion strength value of a class C_i is determined by using the sum of P_{ij} and P_{ji} values in the corresponding row and column of a confusion matrix. The algorithm moves along the row of the class C_i and assigns the largest sum of P_{ij} and P_{ji} value (i≠j) as the confusion strength value of that class (CSV (C_i)). Next, the classes are sorted from largest to smallest according to their CSV values and distributed into groups one

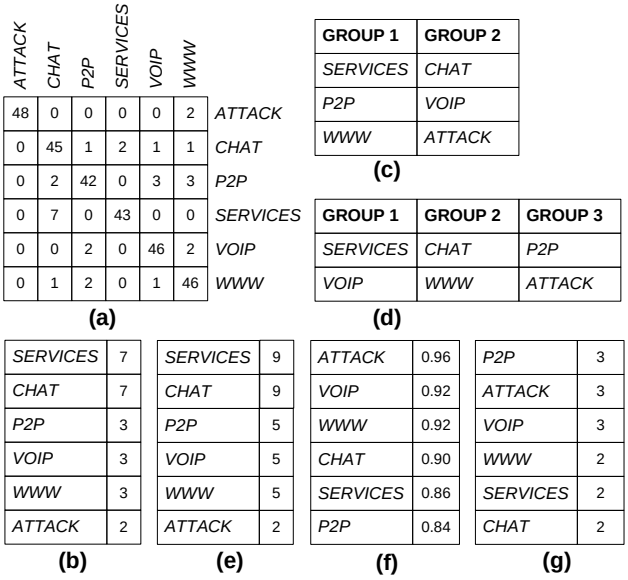

FIGURE 4. a) Confusion matrix b) Sorted class list (r-CMCA) c) Class distribution for 2-groups (r-CMCA) d) Class distribution for 3-groups (r-CMCA) e) Sorted class list (s-CMCA) f) Sorted class list (p-CMCA) f) Sorted class list (LCCA)

by one, starting from the top as similar to r-CMCA algorithm. Note that, if the CSV values of multiple classes are the same, they will rank consecutively in the list and the class with the largest index will come first. The pseudo code of s-CMCA algorithm is the same with r-CMCA algorithm given in Algorithm 1 except the $P[i][j]$ values in lines 9, 10, 13 and 14, which calculate the CSV values, will be replaced by $P[i][j]+P[j][i]$ here. Figure 4e shows the sorted list of classes with s-CMCA algorithm according to their CSV values obtained from the confusion matrix given in Figure 4a.

4.1.3. Percentage based (p-CMCA)

In percentage based (p-CMCA) algorithm, confusion strength value of a class C_i is calculated by the ratio of the P_{ii} value, which gives the number of correct predictions, to the sum of all P_{ij} values in the entire row. All the next steps including CSV based sorting and the grouping are the same as in the previous algorithms described above. The pseudo code of p-CMCA algorithm is given in Algorithm 2. Figure 4f gives the sorted class list output of p-CMCA algorithm for the confusion matrix given in Figure 4a.

In CMCA algorithms, if the number of class metric (n) cannot be exactly divided by the number of groups (G) value given as an external input parameter, the number of classes in some groups may be unbalanced. For this reason, it is recommended to determine the number of groups according to the number of classes for an even distribution. For example, if the number of groups is chosen as *2* or *3* for the number of classes *6*, the sizes of the groups will be equal.

Algorithm 1 r-CMCA algorithm

Input: Confusion matrix file for n classes (CM_File)
Input: Number of groups (G)
Output: Created groups ($Groups[]$)
1: Read from CM_file to confusion matrix ($P[n][n]$) and classes names ($C\_Name[n]$)
2: $i = 1$
3: $C[n].CSV = \{\ 0\ \}$
4: $C[n].Name = \{\ ''\ \}$
5: **while** $i \leq n$ **do**
6: $j = 1$
7: **while** $j \leq n$ **do**
8: **if** $(i \neq j)$ **then**
9: **if** $(C[i].CSV < P[i][j])$ **then**
10: $C[i].CSV = P[i][j]$
11: $C[i].Name = C\_Name[i]$
12: **end if**
13: **if** $(C[j].CSV < P[i][j])$ **then**
14: $C[j].CSV = P[i][j]$
15: $C[j].Name = C\_Name[j]$
16: **end if**
17: **end if**
18: $j = j + 1$
19: **end while**
20: $i = i + 1$
21: **end while**
22: Sort C[n] array by CSV field
23: $Groups[G].CSV = \{\ 0\ \}$
24: $Groups[G].Name = \{\ ''\ \}$
25: $i = 1$
26: **while** $i \leq n$ **do**
27: $j = 1$
28: **while** $j \leq G$ **do**
29: $Groups[j] = C[i]$
30: $i = i + 1$
31: $j = j + 1$
32: **end while**
33: **end while**
34: **return** $Groups[]$

4.2. Leaf count-based class aggregation algorithm (LCCA)

In a decision tree, non-leaf nodes test for a particular feature while comparing an attribute value with a constant whereas leaf-nodes outputs the classification result. Once a leaf node is reached during the search, the sample input is classified according to the class assigned to that leaf. The number of leaf-nodes in the decision tree and the specific class assigned to each are determined by the decision tree algorithm used. In leaf count-based class aggregation algorithm (LCCA), the number of all leaves belonging to a specific class C_i in the decision tree is determined as the leaf count value (LCV) of that class and demonstrated as ($LCV\ (C_i)$). As in other algorithms, all classes are ordered according to their LCV values, and classes are distributed one by one into the group buckets in this order. Even though we do not initially know for sure what the node distribution of the newly created trees will be for each group, we expect the groups to be balanced considering the original tree leaf node distribution. The pseudo code of LCCA algorithm is shown in Algorithm 3. The sorted list of classes with

their *LCV* values obtained from the decision tree in Figure 1a by using LCCA algorithm is presented in Figure 4g. The basic philosophy of all the algorithms aforementioned above is to sort the classes according to a certain parameter (CSV or LCV values) and then distribute the classes to the groups in a balanced way by using this order.

5. OPTIMIZATION

In PDT construction, choosing the appropriate algorithm that provides the best throughput and accuracy results, determining the number of groups (or decision trees) at each stage and deciding on the number of repetitions of the algorithm executions (fixing the levels of PDT) are all optimization issues particularly depending on the data set. We will share our observations in our studies that can help to fix those optimization parameters.

The two important performance criteria, *throughput* and *accuracy*, guide us when making our choices. In hardware implementations, the *throughput* is determined by the system's clock or more specifically the number of classifications performed during a single clock period. The whole search in a PDT is performed within a single clock cycle in parallel once implemented with range comparators by taking advantage of the tree nodes being disjoint as similar to the case demonstrated in Figure 1d (DPRC based implementation). The length of the bit vectors that are determined by the leaf nodes of associated trees dramatically affects the clock period. In PDT architecture design, the overall *throughput* depends on the individual tree having the largest leaf count value among all the trees in the hierarchical structure. Therefore, the *largest leaf count* (LLC) value to be obtained by grouping is our first observation parameter. As the number of levels of the PDT structure increases, it is obvious that the *accuracy* value gradually decreases and the loss will be higher compared to the original decision tree. From this point of view, our goal is to achieve real-time classification with high throughput and low delay by accepting a small loss from the original accuracy as the second observation criteria.

Table 1 shows the classification results obtained from 2-level PDT (separately for 2 ($G\#2$) or 3 ($G\#3$) groups at the Stage 2) created with the grouping algorithms described above using the example decision tree given in Figure 1a and the confusion matrix given in Figure 4a. The leaf count and accuracy values of each tree at each stage are given separately. Furthermore, the LLC *value* and the overall *accuracy*, which are critical metrics to choose the suitable algorithm for the PDT structure are demonstrated separately for each algorithm. In the last row, the results of the original decision tree given

Algorithm 2 p-CMCA algorithm

Input: Confusion matrix file for n classes (CM_File)
Input: Number of groups (G)
Output: Created groups ($Groups[]$)
Read from CM_file to confusion matrix ($P[n][n]$) and classes names ($C\_Name[n]$)
$i = 1$
$Ratio = 0$
$C[n].CSV = \{ 0 \}$
$C[n].Name = \{ \text{' '} \}$
while $i \leq n$ **do**
  $j = 1$
  $T_{row} = 0$
  while $j \leq n$ **do**
    $T_{row} = T_{row} + P[i][j]$
    $j = j + 1$
  end while
  $Ratio = P[i][i]/T_{row} * 100$
  $C[i].CSV = Ratio$
  $C[i].Name = C\_Name[i]$
  $i = i + 1$
end while
Sort C[n] array by CSV field
$Groups[G].CSV = \{ 0 \}$
$Groups[G].Name = \{ \text{' '} \}$
$i = 1$
while $i \leq n$ **do**
  $j = 1$
  while $j \leq G$ **do**
    $Groups[j] = C[i]$
    $i = i + 1$
    $j = j + 1$
  end while
end while
return $Groups[]$

Algorithm 3 LCCA algorithm

Input: n class Decision Tree (DT_File)
Input: Number of groups (G)
Output: Created groups ($Groups[]$)
Create Decision Tree from DT_File and get classes names ($C\_Name[n]$)
$C[n].LCV = \{ 0 \}$
$C[n].Name = \{ \text{' '} \}$
$TravelsalDT\_BFS(DTreeRoot, C[])$
Sort C[n] array by LCV field
$Groups[G].LCV = \{ 0 \}$
$Groups[G].Name = \{ \text{' '} \}$
$i = 1$
while $i \leq n$ **do**
  $j = 1$
  while $j \leq G$ **do**
    $Groups[j] = C[i]$
    $i = i + 1$
    $j = j + 1$
  end while
end while
return $Groups[]$

$TravelsalDT\_BFS(Root, C[])$
if $(Root\ != NULL)$ **then**
  if $(Root.Left == NULL \delta\delta Root.Right == NULL)$ **then**
    $C[Root.Class_index].LCV = C[Root.Class_index].LCV + 1$
  end if
  $TravelsalDT\_BFS(Root.Right, C[])$
  $TravelsalDT\_BFS(Root.Left, C[])$
end if
return $C[]$

TABLE 1. Comparison of class aggregation algorithms using the example decision tree given in Figure 1a and the confusion matrix given in Figure 4a.

CAA	Stage / Group		Leaf Count		Accuracy %	
			G #2	G #3	G #2	G #3
r-CMCA	Stage 1		5	12	88.67	87.33
	Stage 2	Group A	3	4	94.67	94.00
		Group B	6	3	96.67	98.00
		Group C		3		99.00
	Overall		**6**	**12**	**84.83**	**84.71**
s-CMCA	Stage 1		5	8	85.67	92.67
	Stage 2	Group A	5	3	94.00	85.00
		Group B	9	2	96.00	95.00
		Group C		3		99.00
	Overall		**9**	**8**	**81.38**	**86.18**
p-CMCA	Stage 1	Stage 1	7	12	95.67	87.33
	Stage 2	Group A	6	3	90.67	99.00
		Group B	5	4	94.00	94.00
		Group C		3		98.00
	Overall		**7**	**12**	**88.33**	**84.71**
LCCA	Stage 1	Stage 1	5	10	86.33	93.33
	Stage 2	Group A	5	3	94.00	94.00
		Group B	9	5	94.67	94.00
		Group C		2		99.00
	Overall		**9**	**10**	**81.44**	**89.29**
Original C4.5 DT			**15**		**90.00**	

in Figure 1a are also demonstrated. Once the scores for our simple example were examined, the minimum LLC *value* (6) was observed in the PDT ($G\#2$) created by r-CMCA, while the highest accuracy (89.29%) was achieved for the PDT ($G\#3$) constructed using LCCA. However, a PDT ($G\#2$) created with p-CMCA, whose accuracy (88.33%) is the second closest to the original tree (90.00%) and has a lower LLC *value* (7) than LCCA (10), can also be preferred as one of the most suitable alternative. Figure 5 presents the final 2-level PDT data structure with 2-groups ($G\#2$) created by p-CMCA algorithm and bit vector representations of trees. The tree in Stage 1 is a decision tree that distinguishes the two classes Group A (WWW, P2P, Attack) and Group B (Services, VOIP, Attack). The bitmap transformations of unique ranges of F1, F2 and F3 attributes are also shown in Stage 1 on the right. In Stage 2, Group A and B decision trees, each of which has 3 classes, and the bitmap transformations of these trees are shown separately. Once compared to Bit vector coded-DT in Figure 1c, we see that the bitmap length has decreased from 15 to 7 (less than half for the largest tree in Stage 1) while the total number of bitmaps has increased from 25 to 30. The increase in the number of bitmaps corresponds to the increase in the number of parallel comparator units in hardware, but the bit vector length retained in each comparator unit is considerably shortened while providing considerable throughput advantage. The figure also shows the classification steps for example incoming flow information (F1=33879, F2=14, F3=18). The range searches in Stage 1 output 0000001 (bitwise AND of F1 (1001101), F2 (0111011) and F3 (1110111) bitmaps) that selects Group B (corresponds to the rightmost leaf node of root DT). The search of Group B in Stage 2 produces 01000 (bitwise AND of F1 (01000) and F3 (11010) bitmaps) that matches VOIP as the final classification result.

6. ARCHITECTURE AND IMPLEMENTATION ON FPGAS

The structure of PDT (the number of stages and trees in each level) depends on the CAA algorithms and the optimization processes which are expected to give the best throughput and accuracy on the specific data set to be used. The resulting PDT data structure is then embedded in the hardware. We propose Discrete Parallel Range Comparators (DPRC) based architecture to support PDT data structure (DPRC-PDT). Figure 6 presents the 2-stage DPRC-PDT architecture with 2-groups in Stage 2.

Flow level feature values of the traffic flow are given at the input of the classifier and the application class of the flow is determined at the output. Note that we assume that a preceding system that we call *header extracter* calculates the flow level feature values and feeds them to the classifier as in similar studies [10, 45]. Feature values obtained by header extracter are searched in corresponding Feature Tree (FT_i) blocks at each stage. The number of FT_i units in each block depends on the number of attributes in a corresponding decision tree. Here, each FT_i unit representing a feature tree stores the discrete range values of a specific feature i in its parallel range comparator units. A single output is obtained by bitwise AND operation of FT_i outputs specific to each block and those output bitmaps are then combined with a multiplexer to obtain the final classification result. Note that the search operations in FT_i blocks in Stage 1 and Stage 2 are performed simultaneously in parallel within a single clock cycle. Figure 7 shows the internal structure of a single FT_i unit that comprises register pairs to store lower (R_{LOW_i}) and upper (R_{HIGH_i}) boundary values for each disjoint interval, a bit vector register and range comparator units. The incoming search key is compared with the boundary values stored in the registers and the outputs "0" or "1" (falls in a range). Accordingly, the bit vector of matching range is transferred to the output.

We implement DPRC-based PDT architecture on FPGAs platform by utilizing its great ability to run multiple hardware units in parallel. Current FPGA architectures that run pipeline structures generally use Block RAMs (BRAMs) but these units substantially increase clock period due to memory I/O latency. However, in our DPRC-based architecture we only use registers instead of BRAMs, therefore the clock period is much shorter and thus PDT design is proportionally much faster than the classical pipeline architectures.

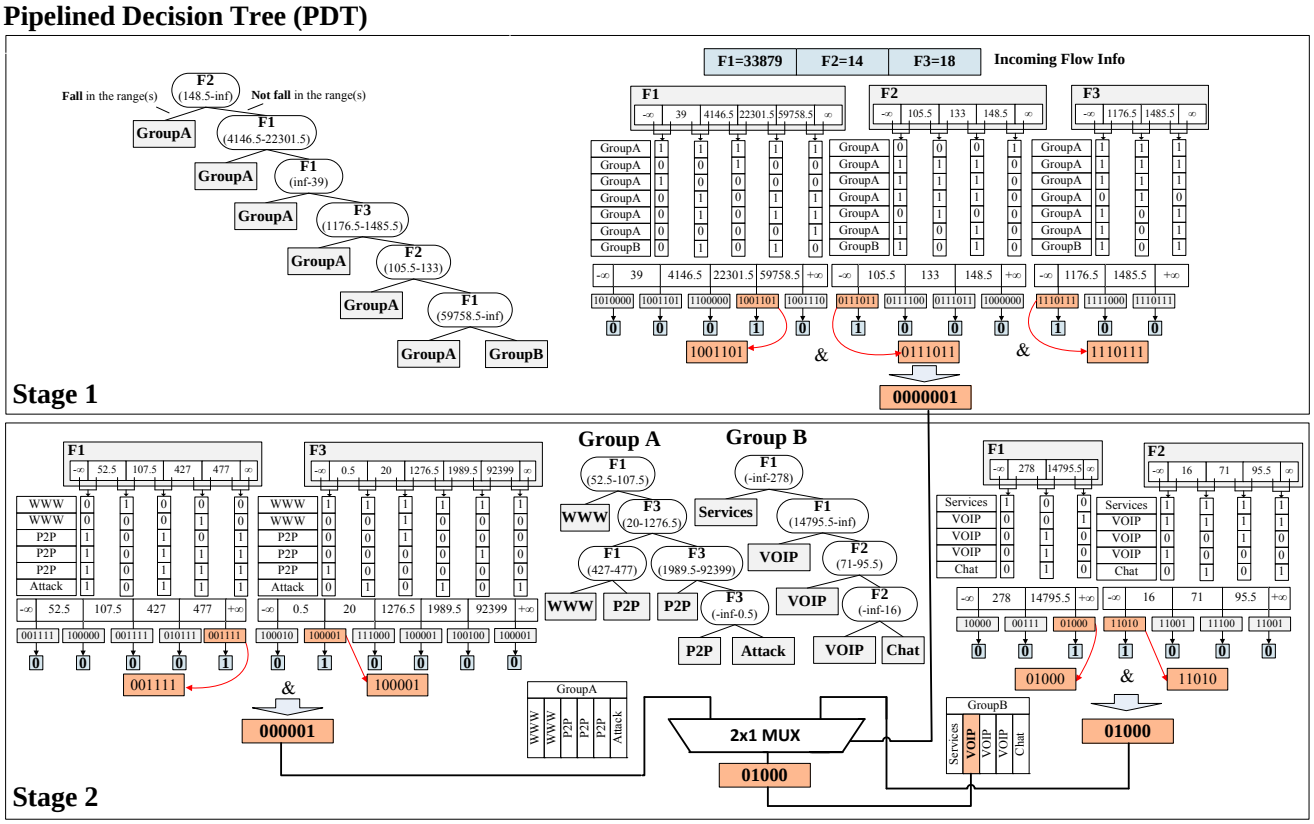

FIGURE 5. 2-level PDT data structure with 2-groups ($G\#2$) created by p-CMCA algorithm

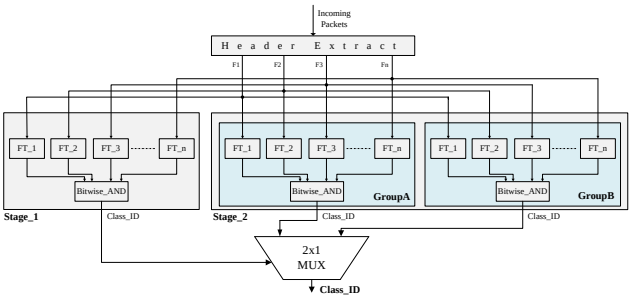

FIGURE 6. 2-stage DPRC-PDT architecture with 2-groups in Stage 2

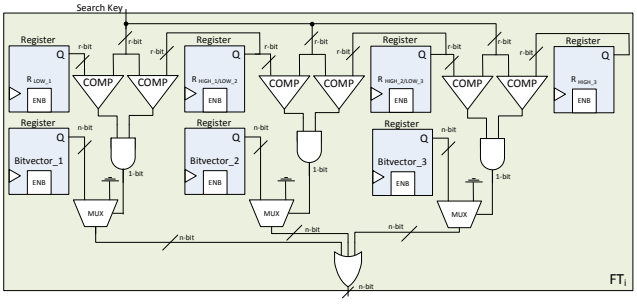

FIGURE 7. A single FT_i unit

7. PERFORMANCE EVALUATION

7.1. Experimental Setup

A traffic flow dataset from [49] comprising 12 application classes with 12 selected features were used in our experiments. Tables 2 and 3 list the application classes and feature set definitions, respectively. Data preparation steps such as feature extraction and selection processes are not covered here because they are beyond the scope of the article. WEKA tool [50], Python and C programming languages were used to preprocess the data and simulate the proposed data structures. Furthermore, Entropy-MDL discretization algorithm [51] is used to convert continuous attribute values to discrete ones.

7.2. Performance Comparison of ML Algorithms

We initially test the performance of popular machine learning algorithms with the data set shown in Table 2. The comparison results in terms of the statistical metrics *Accuracy* (Columns 3), *Kappa* (Columns 4) and *F-Measure* (Columns 5) values are demonstrated in Table 4. The accuracy rate, which shows the ratio of correctly classified data to all data, is the most preferred criterion to measure the success of the ML model. Kappa test, on the other hand, is a statistical method

TABLE 2. Application classes

Class	Application	# flows
ATTACK	Port scans, worms, viruses	750
BULK	FTP, wget	750
DATABASE	MySQL, dbase, Oracle	750
INTERACTIVE	SSH, TELNET, VNC, GotoMyPC	750
MAIL	IMAP, POP, SMTP	750
P2P	Napster, Kazaa, Gnutella, eDonkey	750
SERVICES	X11, DNS, IDENT, LDAP, NTP	750
VOIP	Skype	750
WWW	Web browsers, web applications	750
CHAT	MSN Messenger, Yahoo IM, Jabber	500
MULTIMEDIA	Windows Media Player, Real, iTunes	500
GAMES	Microsoft Direct Play	150
	Total	7900

TABLE 3. Candidate Feature List

Feature	Description
push_pkts_serv	Count of all packets with push bit set in TCP header (server to client)
init_win_bytes_clnt	The total number of bytes sent in initial window (client to server&server to client)
init_win_bytes_serv	The total number of bytes sent in initial window (client to server&server to client)
avg_seg_size_serv	Average segment size: data bytes divided by # packets (server to client)
IP_bytes_med_clnt	Median of total bytes in IP packet (client to server)
act_data_pkt_clnt	Count of packets with at least 1 byte of TCP data payload (client to server)
data_bytes_var_serv	Variance of total bytes in packets (server to client)
min_seg_size_clnt	Minimum segment size observed (client to server)
RTT_samples_clnt	Total numbers of RTT samples found (client to server)
push_pkts_clnt	Count of all packets with push bit set in TCP header (client to server)
serv_port	Server port
clnt_port	Client port

to measure the reliability of agreement between two or more observers (inter-rater reliability). The Kappa coefficient ranges from -1 to $+1$. If the consistency of the observed values is greater than or equal to those by chance, then $K \geq 0$, otherwise $K < 0$. If the Kappa value is between 0 and $+1$, it can be interpreted correctly, but negative ($K < 0$) values are meaningless for reliability. The F score, also known as the F1 score, is a measure of the accuracy of the model in a data set and ranges from 0.0 (worst) to 1.0 (perfect). The performance of the decision tree-based algorithms are also evaluated in terms of tree depth (Column 6), number of leaf nodes (Column 7) and total number of nodes (Column 8).

Once we examine the results, we see that decision tree-based ML algorithms exhibit better accuracy performance with higher Kappa and F-Measure values. In fact, the most critical advantage of decision tree-based algorithms is that the hardware implementations of these structures are feasible and more effective than the other algorithms due to pipeline mapping. However, the main problems with tree structures are their longer tree depths translating into longer delays due to unbalanced node distribution and the excessive growth of tree sizes with the increase in the number of application classes that cause substantial performance degradation in the hardware as stated earlier. We overcome this issue and provide scalability against the expected fast growth of the number of applications while sacrificing some loss of accuracy due to the multiple classification stages. However, alternative optimizations together with the appropriate CAA algorithms are possible to minimize the loss in accuracy.

7.3. Performance analysis of CAA algorithms

We test the performance of our proposed PDT architecture in terms of *throuhput* and *accuracy*, using CAA algorithms with the data set summarized in Tables 2. Accuracy value mainly depends on the chosen machine learning algorithm. In all analyzes in this study, C4.5 Decision Tree algorithm, which gives the highest accuracy score according to the results in Table 4 were used. PDT is a multi-level data structure consisting of one or more small C4.5 decision trees at each stage as shown in Figure 2. The fact is that, if we increase the number of stages in the PDT structure, the size of the trees at each level decrease relatively. Therefore, as the number of application classes increases in the future, the need to increase the number of stages will be inevitable in order not to experience throughput loss due to higher leaf count values. On the other hand, once the number of levels in PDT increases, we lose from accuracy as the misclassifications spread to the trees below. Therefore, we need to consider the trade off between *accuracy* and *throughput* when determining the number of stages. Since the number of application classes in our dataset is relatively low (only 12 classes), we performed our analyzes on the 2-stage PDT structure. Another important criterion is the number of groups (or trees) at each level, and this value is given as an input parameter to CAA algorithms. The leaf count (LC) values of the trees for the various options of 2, 3, 4, 5 and 6 groups (in Stage-2) created with each CAA algorithm and the overall accuracy values of the resulting PDT structure are presented in Table 5.

We know that the LLC *value* in the PDT structure determines the clock time or implicitly the throughput. According to the Table 5, we see that LLC *value* in all algorithms is determined by the root decision tree in Stage-1 (Column 3), but this may not always be the case. We also observed that the leaf count value generally increases (despite some exceptions such as r-CMCA and s-CMCA Group 5) as the number of groups increased. Therefore, we may conclude that more balanced trees can be obtained by increasing the

TABLE 4. Performance Comparison of Machine Learning Algorithms

Algorithm	Type	Accuracy (%)	Kappa	F-Measure	Tree Depth	# of leaf nodes	Total # of nodes
C4.5 [53]	Decision Tree	97.0100	0.959	0.962	37	164	327
SimpleCART [26]	Decision Tree	96.4810	0.961	0.965	17	106	211
BF Tree [52]	Decision Tree	96.4937	0.962	0.965	17	88	175
RIPPER [57]	Rule-Based	96.0380	0.957	0.960	NA	NA	NA
k-NN [54]	Distance-based	95.1772	0.947	0.952	NA	NA	NA
BayesNET [55]	Statistical	92.6835	0.920	0.927	NA	NA	NA
Naive Bayes [56]	Statistical	92.2152	0.915	0.923	NA	NA	NA
ANN [58]	Function-Based	79.5949	0.776	0.797	NA	NA	NA

TABLE 5. Performance Analysis of CAA algorithms (LC:Leaf Count, St.:Stage)

CAA	Group	St.1 (LC)	St.2 (LC)						Acc. %
			A	B	C	D	E	F	
r-CMCA	2	71	39	36					95.71
	3	107	5	26	12				95.64
	4	113	3	9	16	11			96.42
	5	109	2	3	12	13	4		95.75
	6	122	2	3	5	6	4	4	95.96
s-CMCA	2	60	41	32					96.04
	3	80	19	13	17				96.11
	4	128	11	6	7	17			95.54
	5	98	11	6	4	4	4		96.08
	6	131	5	2	4	4	4	3	95.91
p-CMCA	2	58	46	44					95.23
	3	83	11	27	18				96.20
	4	80	3	9	10	17			95.98
	5	108	7	3	4	10	5		96.28
	6	120	2	3	4	4	5	2	96.46
LCCA	2	66	47	43					95.98
	3	98	25	13	13				95.54
	4	100	6	17	8	9			96.04
	5	111	2	9	5	10	16		96.24
	6	123	2	3	2	2	4	7	96.34
Original DT		**164**							**97.01**

number of levels rather than increasing the number of groups. As seen in the table, the best LLC *value* (58) which is almost one third of the leaf count value of the original C4.5 decision tree (164) is reached by p-CMCA algorithm for 2-groups but the accuracy value decreased from 97.01% in the original tree to 95.23%. On the other hand, with LLC *value* (60) very close to p-CMCA, higher accuracy 96.04% is achieved by the s-CMCA algorithm as can be seen in the last column of Table 5. As a result, s-CMCA algorithm with group number 2 is preferred for the memory and throughput analysis of PDT architecture in the following sub-sections.

7.4. Memory Requirement

PDT data structure is a set of decision trees at multiple levels. Each decision tree consists of multiple feature trees (FT_i) representing separate attributes of that tree. There are as many FT_i as the number of attributes in the tree. Each node of an FT_i tree stores disjoint feature interval values and bit vectors. As stated before, we implement FT_i trees with parallel range comparator units in hardware (DPRC-PDT). The range boundary values are kept in registers and the number of registers for each FT_i is equal to the number of intervals (or the number of nodes) in that tree. The first row of Table 6 gives the required register size, in bits, to store the largest interval boundary value for each attribute F_k ($1 \leq k \leq 12$). This values are specific to each attribute. For example, the largest value that can be stored in the 16-bit length register of F_1 attribute is 65535 but this value is 3 for 2-bit length register assigned for F_3. An additional register is needed to store the bit vector value and the length of bit vector register is equal to the leaf count value of the corresponding tree. In Table 6, the leaf count values of each distinct tree structures are given for 2-stage PDT architecture with 2-groups in Stage 2 (Stage-2A and 2B) and the single stage bit vector coded decision tree (BC-DT) as a baseline model separately. The memory requirement of the PDT structure is simply calculated by multiplying the number of registers used and their sizes. The number of registers recording the boundary values are also given in Table 6 separately for each attribute F_k. Similarly, the size of the registers that store the bit vectors is determined with the leaf count value given for each tree. As a result, the total memory requirement of the PDT and BC-DT architectures are given in the last column of Table 6 for comparison purposes. Note also that DPRC-based implementations save more memory than pipeline implementations because of the elimination of pointer fields in tree nodes. Finally, we conclude that the memory requirement of the PDT architecture is almost half of the BC-DT where both architectures are implemented with parallel range comparator units.

7.5. Throughput

We implemented the proposed PDT architecture on Field Programmable Gate Arrays (FPGAs) using Verilog on the Xilinx Vivado 2020.2 platform. Xilinx Virtex-Ultra XCVU440FLGA2892 with speed range of -3 were chosen as the target device. Figure 8 represents the clock period change of the BC-DT and PDT architectures with respect to the increasing number of classes. From the graph, we see that the clock period of the BC-DT architecture increases very rapidly as the number of application classes increases, but the PDT architecture exhibits a very slight increase in the same scenario. We conclude that PDT architecture

TABLE 6. Memory Requirement of PDT structure

Architecture		Size (Bits)	Feature Set F1 16	F2 16	F3 2	F4 2	F5 2	F6 11	F7 11	F8 12	F9 12	F10 2	F11 10	F12 10	Leaf Count	Memory (Bytes)
BC-DT	Single Stage	Number of Range Intervals	56	28	3	3	3	50	49	53	50	3	17	62	164	8288.88
PDT	Stage-1		30	21	2	2	2	23	31	23	25	2	9	19	60	
	Stage-2A		31	26	3	3	3	27	22	28	24	2	12	32	41	4193.50
	Stage-2B		25	22	3	3	3	25	25	27	25	3	8	29	32	

TABLE 7. Implementation Results

Architecture	Throughput (Gbps)	Throughput (MCPS)	Clock (ns)	Frequency (MHz)	Number of LUTs / Total	Number of FFs / Total
BC-DT	721.70	2255.30	4.43	225.53	5530 / 2532960 (0.2183%)	2376 / 5065920 (0.0469%)
PDT	928.88	2902.76	3.45	290.28	7982 / 2532960 (0.3151%)	1880 / 5065920 (0.0371%)

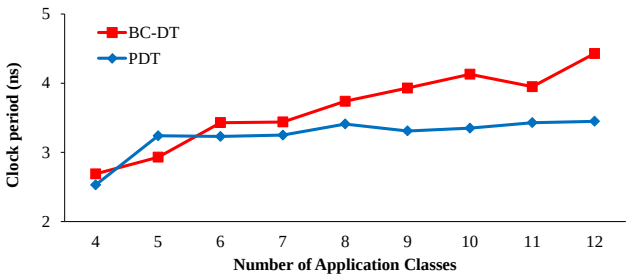

FIGURE 8. Clock period change of PDT architecture depending on the number of classes

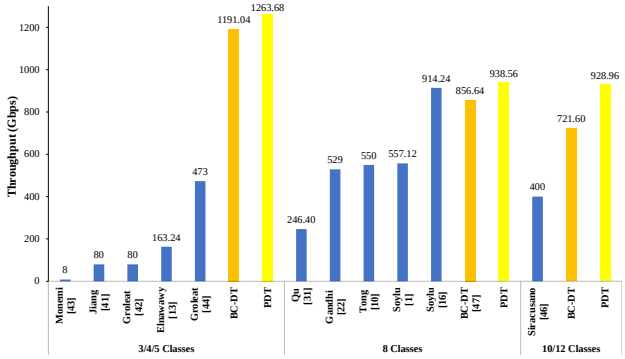

FIGURE 9. Throughput comparison of ML-based traffic classification engines implemented on FPGAs

emerges as a scalable solution against the expected fast growth of the number of classes in the future. Table 7 presents the Place and Route FPGA implementation results of the BC-DT and proposed PDT architectures with 12 application classes. In the throughput results (Gbps), the packet size is considered as the minimum value of 40 Bytes (or 320 bits). The results confirm that the throughput rate of PDT scheme (2902.76 million classifications per second (MCPS)) is about 29% higher than that of baseline BC-DT architecture (2255.3 MCPS). On the other hand, we can also state absolutely that as the number of classes increases, the gap between the throughput performances of the those architectures will get larger even more.

Additionally, Figure 9 shows the throughput performance (in Gbps) comparison of ML-based popular traffic classification architectures in the literature. To be fair in the comparison, the number of classes in PDT and BC-DT architectures were balanced with those given in the existing studies, and thus the additional results for the case of 3/4 and 8 classes are also included in the figure. The results prove that PDT scheme outperforms of all the related studies in all cases. Furthermore, the range merging optimization process as we employed in our previous study (DPRC-based BC$-$SC^{opt} in [16]) is skipped here and not included within our results for simplicity. Note also that, the range merge optimized performance result of the architecture proposed in [16] is given in the figure, even the PDT without optimization performs better than this classifier. It is clear that if the same optimization process were added to the PDT structure, the improvement would be much more.

8. CONCLUSION

The design of high speed traffic classifiers that can handle packet processing in link rates has been a major challenge and requires intensive investigations from the researchers to be able to cope with the rapid growth of the Internet as well as advances in optical networking technology. In this paper, a novel and scalable PDT traffic classification engine and class aggregation approaches (LCCA and CMCA with its variants) that can handle all the negative impacts of application class increase on the throughput and latency performance of decision tree based hardware classifiers are proposed. We developed an adaptive multi-level data structure consisting of multiple decision trees at each stage where the number of stages and also the size of those trees can be optimized to achieve the best scores in *throughput* and *accuracy* metrics. We further transform multi-trees into bit vectors and map onto FPGA-based hardware utilizing from parallel range comparator units. Experimental results confirm that the PDT engine outperforms all relevant designs and the improvements will be much larger over competitors

as the number of classes increases in forthcoming periods. As a future work, we aim to develop an optimization model that can estimate the number of stages and also the number of trees in PDT at the beginning to ensure the better management of performance aspect in all various circumstances.

DATA AVAILABILITY STATEMENT

The datasets provided by [49] are available at http://www.cl.cam.ac.uk/research/srg/netos/brasil/.

REFERENCES

[1] Soylu, T., Erdem, O., Carus, A. and Güner, E.S. (2017) Simple CART based real-time traffic classification engine on FPGAs. *Proceedings of ReConFig 17*, Cancun, Mexico, 04-06 December, pp. 1-8. IEEE, New York.

[2] Karagiannis, T., Broido, A., Brownlee, N. and Claffy, K. (2004) Is P2P dying or just hiding?. *Proceedings of Globecom 04*, Dallas, TX, USA, 29 November-03 December, pp. 1532-1538. IEEE, New York.

[3] Harthi, A.F.A. (2015) Designing an accurate and efficient classification approach for network traffic monitoring (Doctor of Philosophy), RMIT University, Melbourne, Australia

[4] Hubballi, N., Swarnkar, M., and Conti, M. (2020) BitProb: Probabilistic Bit Signatures for Accurate Application Identification. *IEEE Transactions on Network and Service Management*, 17, 1730-1741.

[5] Hubballi, N. and Khandait, P. (2022) KeyClass: Efficient keyword matching for network traffic classification. *Computer Communications*, 185, 79-91.

[6] Bu, Z., Zhou, B., Cheng, P., Zhang, K. and Ling, Z.H. (2020) Encrypted network traffic classification using deep and parallel network-in-network models. *IEEE Access*, 8, 132950-132959.

[7] Zhao, J., Jing, X., Yan, Z. and Pedrycz, W. (2021) Network traffic classification for data fusion: A survey. *Information Fusion*, 72, 22-47.

[8] Shen, M., Ye, K., Liu, X., Zhu, L., Kang, J., Yu, S., Li, Q., and Xu, K. (2022) Machine Learning-Powered Encrypted Network Traffic Analysis: A Comprehensive Survey. *IEEE Com. Surveys and Tutorials*, doi: 10.1109/COMST.2022.3208196.

[9] Nguyen, T.T. and Armitage, G. (2008) A survey of techniques for internet traffic Classification using machine learning. *IEEE Com. Surveys and Tutorials*, 10, 56-76.

[10] Tong, D., Sun, L., Matam, K. and Prasanna, V.K. (2013) High throughput and programmable online traffic classifier on FPGA. *Proceedings of FPGA 13*, Monterey, California, USA, 11-13 February pp. 255-264. ACM, New York.

[11] Qu, Y.R. and Prasanna, V.K. (2015) Enabling high throughput and virtualization for traffic classification on FPGA. *Proceedings of FCCM 15*, Vancouver, BC, Canada, 2-6 May, pp. 44-51. IEEE, New York.

[12] Pacheco, F., Exposito, E., Gineste, M., Baudoin, C., and Aguilar, J. (2019) Towards the Deployment of Machine Learning Solutions in Network Traffic Classification: A Systematic Survey. *IEEE Com. Surveys and Tutorials*, 21, 1988-2014.

[13] Elnawawy, M., Sagahyroon, A. and Shanableh, T. (2020) FPGA-based network traffic classification using machine learning. *IEEE Access*, 8, 175637-175650.

[14] Salman, O., Elhajj, I. H., Kayssi, A. and Chehab, A. (2020) A review on machine learning-based approaches for Internet traffic classification. *Annals of Telecommunications*, 75, 673-710.

[15] Chen, J., Breen, J., Phillips, J.M. and Merwe, J.V. (2022) Practical and configurable network traffic classification using probabilistic machine learning. *Cluster Computing*, 25, 2839-2853.

[16] Soylu, T., Erdem, O. and Carus, A. (2020) Bit vector-coded simple CART structure for low latency traffic classification on FPGAs. *Computer Networks*, 167, 106977.

[17] Khatouni, A. S., Seddigh, N., Nandy, B. and Heywood, N. Z. (2021) Machine Learning Based Classification Accuracy of Encrypted Service Channels: Analysis of Various Factors. *Journal of Network and Systems Management*, 29, 8.

[18] Tahaei, H., Afifi, F., Asemi, A., Zaki, F. and Anuar, N. B. (2020) The rise of traffic classification in IoT networks: A survey. *Journal of Network and Computer Applications*, 154, 102538.

[19] Kornaros, G. (2022) Hardware-Assisted Machine Learning in Resource-Constrained IoT Environments for Security: Review and Future Prospective. *IEEE Access*, 10, 58603-58622.

[20] Bout, E., Loscri, V., and Gallais, A. (2022) How Machine Learning Changes the Nature of Cyberattacks on IoT Networks: A Survey. *IEEE Com. Surveys and Tutorials*, 24, 248-279.

[21] Wang, H., Qu, Z., Zhou, Q., Zhang, H., Luo, B., Xu, W., Guo, S., and Li, R. (2022) A Comprehensive Survey on Training Acceleration for Large Machine Learning Models in IoT. *IEEE Internet of Things Journal*, 9, 939-963.

[22] Gandhi, V.R., Qu, Y.R. and Prasanna, V.K. (2014) High-throughput hash-based online traffic classification engines on FPGA. *Proceedings of ReConFig 14*, Cancun, Mexico, 8-10 December pp. 1-6. IEEE, New York.

[23] Kim, H., Claffy, K., Fomenkov, M., Barman, D., Faloutsos, M. and Lee, K. (2008) Internet traffic classification demystified: Myths, caveats, and the best practices. *Proceedings of ACM CoNEXT 08*, Madrid, SPAIN, 10-12 December, pp. 1-11. ACM, New York.

[24] Alshammari, R. and Zincir-Heywood, A.N. (2009) Machine learning based encrypted traffic classification: Identifying ssh and Skype. *Proceedings of IEEE CISDA 09*, Ottawa, ON, Canada, 08-10 July, pp. 289-296. IEEE, New York.

[25] Monemi, A., Zarei, R. and Marsono, M. N. (2013) Online NetFPGA decision tree statistical traffic classifier. *Computer Communications*, 36, 1329-1340.

[26] Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J. (1984) Classification and regression trees. *Wadsworth Publishing Co.*. Chapman and Hall/CRC, Florida, ABD.

[27] Silver, B. (1990) Netman: A learning network traffic controller. *Proceedings of IEA/AIE 90*, Charleston South Carolina, USA, pp. 923-931. ACM, New York.

[28] Frank, J. (1994) Artificial Intelligence and intrusion detection: Current and future directions. *Proceedings of the 17th Computer Security Conference*, Maryland, Washington, D.C., USA, 11-14 October, pp. 923-931. NIST, Gaithersburg.

[29] Este, A., Gringoli, F. and Salgarelli, L. (2009) Support vector machines for TCP traffic Classification. *Computer Networks*, 53, 2476-2490.

[30] Lim, Y.S., Kim, H.C., Jeong, J., Kim, C.K., Kwon, T.T. and Choi, Y. (2010) Internet traffic classification demystified: On the sources of the discriminative power. *Proceedings of ACM Co-NEXT 10*, Philadelphia, USA, 30 Nov.-3 Dec pp. 1-12. ACM, New York.

[31] Qu, Y.R. and Prasanna, V.K. (2014) Compact hash tables for high-performance traffic classification on multi-core processors. *Proceedings of SBAC-PAD 26*, Paris, France, 04 December, pp. 17-24. IEEE, New York.

[32] Caicedo-Munoz, J.A., Espino, A.L., Corrales, J.C. and Rendn, A. (2018) Qos-classifier for vpn and non-vpn traffic based on time-related features. *Computer Networks*, 144, 271-279.

[33] Dias, K. L., Pongelupe, M. A., Caminhas, W. M. and Errico, L. (2019) An innovative approach for real-time network traffic classification. *Computer Networks*, 158, 143-157.

[34] Labayen, V., Magana, E., Morato D. and Izal, M. (2020) Online classification of user activities using machine learning on network traffic. *Computer Networks*, 181, 107557.

[35] Dong, S. (2021) Multi class SVM algorithm with active learning for network traffic classification. *Expert Systems with Applications*, 176, 114885.

[36] Afuwape, A. A., Xu, Y., Anajemba, J. H. and Srivastava, G. (2021) Performance evaluation of secured network traffic classification using a machine learning approach. *Computer Standards and Interfaces*, 78, 103545.

[37] Obasi, T. and Shafiq, M. O. (2022) CARD-B: A stacked ensemble learning technique for classification of encrypted network traffic. *Computer Communications*, 190, 110-125.

[38] Bovenzi, G., Cerasuolo, F., Montieri, A., Nascita, A., Persico, V. and Pescape, A. (2022) A Comparison of Machine and Deep Learning Models for Detection and Classification of Android Malware Traffic. *Proceedings of ISCC 22*, Rhodes, Greece, 30 June - 3 July, pp. 1-6. IEEE, New York.

[39] Nsaif, M., Kovasznai, G., Abboosh, M., Malik, A. and Frein, R. D. (2022) ML-Based Online Traffic Classification for SDNs. *Proceedings of CITDS 22*, Debrecen, Hungary, 16 - 18 May, pp. 217-222. IEEE, New York.

[40] Luo, Y., Xiang, K. and Li, S. (2008) Acceleration of decision tree searching for IP traffic classification. *Proceedings of ANCS 08*, San Jose California, USA, 6 - 7 November, pp. 40-49. ACM, New York.

[41] Jiang, W. and Gokhale, M. (2010) Real-time classification of multimedia traffic using FPGA. *Proceedings of FPL 10*, Milan, Italy, 31 August-02 September, pp. 56-63. IEEE, New York.

[42] Groleat, T., Arzel, M. and Vaton, S. , (2012) Hardware acceleration of SVM based traffic classification on FPGA. *Proceedings of IWCMC*, Limassol, Cyprus, 27-31 August, pp. 443-449. IEEE, New York.

[43] Monemi, A., Zarei, R., Marsono, M. and Khalil-Hani, M. (2013) Parameterizable decision tree classifier on NetFPGA. *Intelligent Informatics*, 182, 119-128.

[44] Groleat, T., Arzel, M., and Vaton, S. (2014) Stretching the Edges of SVM Traffic Classification With FPGA Acceleration *IEEE Transactions on Network and Service Management*, 11, 278-291.

[45] Tong, D., Qu, Y.R. and Prasanna, V.K. (2017) Accelerating decision tree based traffic classification on FPGA and multicore platforms. *IEEE Transactions on Parallel and Distributed Systems*, 28, 3046-3059.

[46] Siracusano, G., Galea, S., Sanvito, D., Malekzadeh, M., Antichi, G., Costa, P., Haddadi, H. and Bifulco, R. (2022) Re-architecting Traffic Analysis with Neural Network Interface Cards. *Proceedings of NSDI 2022*, Renton, WA, USA, 4-6 April, pp. 513-533. USENIX Association.

[47] Soylu, T., Erdem, O., Carus, A. and Güner, E.S. (2018) Real-Time Traffic classification using simple CART forest on FPGAs. *Proceedings of HPSR 2018*, Bucharest, Romania,18-20 June, pp. 1-8. IEEE, New York.

[48] Witten, I.H., Frank, E., Hall, M.A. and Pal, C.J. (2017) Data mining practical machine learning tools and techniques. The Morgan Kaufmann, Massachusetts, USA.

[49] Li, W., Canini, M., Moore, A. W., Bolla, R. (2009) Efficient application identification and the temporal and spatial stability of classification schema. *Computer Networks*. 53, 790-809.

[50] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. and Witten, I.H. (2009) The Weka data mining software: an Update. *SIGKDD Explor. Newsl.*, 11, 10-18.

[51] Fayyad, U.M. and Irani, K.B. (1991) Multi-interval discretization of continuous valued attributes for classification learning. *Proceedings of IJCAI*, Chambery, France, 28 August-3 September, pp. 1022-1027. The Morgan Kaufmann, Massachusetts, USA.

[52] Haijian, S. (2007) Best-first decision tree learning. Master thesis in University of Waikato, Hamilton, New Zeland.

[53] Quinlan, J.R. (1993) C4.5:Programs for machine learning.The Morgan Kaufmann, Massachusetts, USA.

[54] Aha, D. and Kibler, D. (1991) Instance-based learning algorithms. *Machine Learning*, 6, 37-66.

[55] Heckerman, D., Mamdani, A. and Wellman, M.F. (1995) Real-world applications of Bayesian Networks. *Communications of the ACM*, 38, 24-68.

[56] John, G.H. and Langley, P. (1995) Estimating continuous distributions in Bayesian Classifiers. *Proceedings of UAI*, Montreal Que, Canada, 18-20 August, pp. 338-345. San Francisco, CA, USA.

[57] William, W.C. (1995) Fast effective rule induction. *Proc. Machine Learning Proceedings 1995*, Tahoe City, California, 9-12 July, pp. 115-123. The Morgan Kaufmann, Massachusetts, USA.

[58] Hopfield, J.J. (1982) Neural networks and physical systems with emergent collective computational abilities *National Academy of Sciences of the USA*, 79, 2554-2558.