Accelerating Decision Tree Based Traffic Classification on FPGA and Multicore Platforms

Da Tong, *Member, IEEE*, Yun Rock Qu, *Member, IEEE*, and Viktor K. Prasanna, *Fellow, IEEE*

Abstract—Machine learning (ML) algorithms have been shown to be effective in classifying a broad range of applications in the Internet traffic. In this paper, we propose algorithms and architectures to realize online traffic classification using flow level features. First, we develop a traffic classifier based on C4.5 decision tree algorithm and Entropy-MDL (Minimum Description Length) discretization algorithm. It achieves an overall accuracy of 97.92 percent for classifying eight major applications. Next we propose approaches to accelerate the classifier on FPGA (Field Programmable Gate Array) and multicore platforms. We optimize the original classifier by merging it with discretization. Our implementation of this optimized decision tree achieves 7500+ Million Classifications Per Second (MCPS) on a state-of-the-art FPGA platform and 75-150 MCPS on two state-of-the-art multicore platforms. We also propose a divide and conquer approach to handle imbalanced decision trees. Our implementation of the divide-and-conquer approach achieves 10,000+ MCPS on a state-of-the-art FPGA platform and 130-340 MCPS on two state-of-the-art multicore platforms. We conduct extensive experiments on both platforms for various application scenarios to compare the two approaches.

Index Terms—Traffic classification, machine learning, decision tree, multicore, FPGA, high throughput

1 Introduction

Traffic classification [1], i.e., detecting the application of TCP flows, serves as one of the major applications in the network data plane. It benefits many value-added services as well as network security. The rapid growth of the Internet requires traffic classifiers to support extremely high throughput of over hundreds of gigabits per second. However, most of the existing traffic classification engines only support a few tens of gigabits per second throughput [2]; this makes traffic classification a performance bottleneck when they are employed in high speed routers.

In general, existing traffic classification approaches can be categorized into 4 classes: (1) *port number based* schemes classify traffic based on transport layer port numbers. Many applications today assign port numbers dynamically; this means the port number based approaches are no longer reliable. (2) *Deep Packet Inspection* (DPI) compares the traffic payload with known signatures. Although the DPI based techniques [3] can achieve the highest accuracy, they incur long processing latency since the entire payload has to be examined. (3) *heuristic based* techniques [4] classify traffic based on heuristic patterns; compared to other techniques, the classification accuracy of heuristic based techniques is relatively low. (4) *Machine Learning* (ML) techniques, including the well known C4.5 decision tree, examine the statistical properties of the network traffic [5], [6]. ML based techniques have demonstrated higher classification accuracy than other approaches [5], [6]; however, extensive study is still required to select the appropriate set of traffic features that is easy to use and leads to very high classification accuracy.

FPGAs have been widely used to accelerate various types of applications [7], [8], [9], [10], [11], [12]. Meanwhile, a new trend in network applications is to use software accelerators and virtual machines [13], [14], [15]. State-of-the-art multicore processors [16], [17] exploit caches and instruction level parallelism (ILP) to improve the performance; this makes multicore processors an attractive platform for high performance network applications.

In this paper, we first present two algorithms for Internet traffic classification based on carefully selected traffic features. Then we compare the performance of these two algorithms on FPGA and multicore processors. Our contributions include the following:

- We identify a feature set that achieves high classification accuracy through extensive experiments. Our C4.5 decision tree based classifier built upon this flow-level feature set can achieve an overall accuracy of 97.92 percent in classifying eight major network applications.
- We propose two algorithms to accelerate the proposed traffic classifier. The first algorithm exploits an optimized decision tree, while the second algorithm exploits divide and conquer technique that flattens the decision tree into multiple range trees.
- We prototype our approaches on both state-of-the-art FPGA and AMD/Intel multicore platforms.

- *D. Tong is with VMware, Inc., 3401 Hillview Ave, Palo Alto, CA 94304. E-mail: tongd@vmware.com.*
- *Y. R. Qu is with Xilinx, Inc., 2100 Logic Dr, San Jose, CA 95124. E-mail: yunq@xilinx.com.*
- *V. K. Prasanna is with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089. E-mail: prasanna@usc.edu.*

Manuscript received 7 Oct. 2016; revised 5 Apr. 2017; accepted 20 Apr. 2017. Date of publication 12 June 2017; date of current version 11 Oct. 2017. (Corresponding author: Da Tong.)
Recommended for acceptance by M. Huebner.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2017.2714661

Experimental results for the optimized decision trees show 7500+, MCPS throughput on FPGA, and 75-150 MCPS throughput on multicore platforms. Experimental results for the divide and conquer approach show 10000+, MCPS throughput on FPGA, and 130-340 MCPS throughput on multicore platforms.

- We conduct extensive experiments to compare our proposed approaches for various application scenarios on both FPGA and multicore platforms.

The rest of the paper is organized as follows: Section 2 introduces the related work. Section 3 defines the problem. We introduce the training of our traffic classification engines in Section 4. We present the algorithms and architectures for our classification engines on FPGA and multicore processors in Section 5. Section 6 evaluates the performance and Section 7 concludes this paper.

2 Background

In this section, we will introduce the prior work related to the machine learning based traffic classification in Section 2.1; we will present existing acceleration techniques for traffic classification in Section 2.2.

2.1 ML Based Algorithms for Traffic Classification

ML based techniques [1], [5], [6] only explore the property of the traffic flow reflected by the packet headers to make decisions. These properties are referred to as *features*. A feature can be a *packet level* feature (a characteristic of a single packet, e.g., packet size), or a *flow level* feature (a characteristic of a packet flow by some definition, e.g., the average packet size of the packet flow sharing the same 5-tuple information). Some features are both packet level and flow level, for example, IP addresses.

Many ML based algorithms have been proposed for high performance traffic classification, including Naive Bayesian [18], K-means [19], Support Vector Machine (SVM) [20], [21], and comprehensive techniques combining both supervised and unsupervised machine learning algorithms [22], [23]. Among these algorithms, C4.5 decision tree based algorithm [5] has demonstrated the highest classification accuracy in most of the experiments. The effectiveness of various ML based algorithms for classifying encrypted network traffic is studied in [5]. A set of algorithms including Naive Bayesian, SVM, and C4.5 algorithm are evaluated. A total number of 22 features, including both the packet level features (e.g., packet size) and the flow level features (e.g., packet inter arrival time) are used to build the ML based classifiers. Experimental results show that, C4.5 algorithm gives the best accuracy ($>$ 83 percent accuracy for SSH traffic and $>$ 97.8 percent accuracy for Skype traffic) among all the considered techniques. In [6], 9 flow level feature sets are evaluated; packet size, port number and their related statistics are shown to achieve the highest accuracy, regardless of the underlying algorithms that have been used. A discretization process on the feature values is also proposed to improve the accuracy of the classifiers. In [22], 14 NetFlow attributes are used to classify 15 Internet applications. K-means clustering (unsupervised learning) is used to further derive each application's traffic into sub-classes. C5.0 decision tree (supervised learning) is used to build the classifier for the derived subclasses. Combining the contributions from both K-means clustering and the C5.0 decision tree, the classifier achieves an average accuracy of 96.67 percent. In [23], the authors target a situation in which training data is insufficient to cover all the applications. K-means clustering algorithm is used to group the training data consisting of labeled and unknown flows according to 20 statistical features. Instead of being individually classified, the traffic flows are classified in groups by majority voting. The classifier achieves higher F-measure than the other five state-of-the-art machine learning based traffic classification techniques in their experiments.

2.2 Existing Traffic Classifiers

Many existing works have proposed accelerators for traffic classification. In [2], an FPGA based architecture is presented for the C4.5 algorithm; explicit range match is explored and memory accesses are parallelized to improve the performance of their architecture. They use the number of memory accesses instead of throughput as a performance metric. No post-place-and-route results are reported for their implementation on FPGA. In [7], an FPGA based architecture is proposed for multimedia traffic classification. The classifier is based on k-Nearest-Neighbor algorithm and packet level features. Their classifier achieves high accuracy for large training data sets. However, their approach is restricted to classifiers with a small number of applications, and susceptible to noise in the training data. In [24], an SVM based traffic classifier is proposed on NetFPGA platform. The SVM algorithm is parallelized through loop unrolling leveraging the parallelism of NetFPGA. The highest throughput the classifier achieves is 28.6 Gbps. In [25], the decision tree based classifier is converted to a multi-column rule set table, then accelerated by a two dimensional pipelined architecture. By applying column striding and row clustering, the architecture can achieve 588-770 million classifications per second for various synthetic rule set tables. However, the performance results based on real data sets are still missing. In [26], an SVM based approach is explored on a dual Xeon PC; this machine has a total number of 24 cores running at 2.6 GHz and 48 GB RAM. An SVM based algorithm is also employed in [27], targeting a dual Xeon PC with 24 GB RAM and a total number of 24 cores each running at 2.6 GHz. However, these approaches can only achieve 7 MCPS throughput.

3 Problem Definition

Since the C4.5 decision tree algorithm demonstrates the highest classification accuracy in previous studies (as discussed in Section 2.1), we focus our work on the C4.5 decision tree based approach [5], [28]. Our goal is to build a flow level online traffic classifier based on C4.5 decision tree that achieves both high classification accuracy and high classification throughput. Here, we define traffic flow as a series of packets sharing the same 5-tuple information: transport layer protocol, source and destination port numbers, source and destination IP addresses. The input to the classifier is the flow level feature values of the traffic flow (discussed in detail in Section 4.2.4) and the output is the application of the input flow. We assume that a preceding system computes the flow level feature values and feeds them to the classifier.

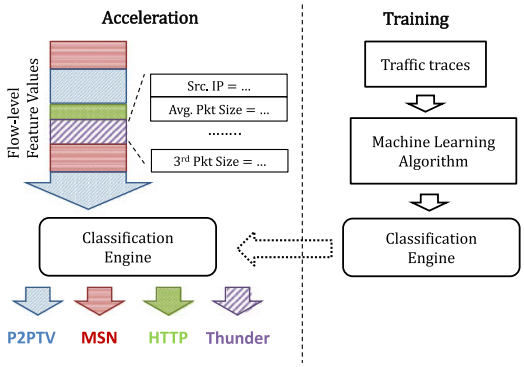


Fig. 1. Problem definition.

To achieve the goal, we target the following 2 problems as illustrated in Fig. 1.

- *Training*: Given the traffic traces and a set of target application class, construct a C4.5 decision tree that most effectively distinguishes the target applications from each other.
- *Acceleration*: Given the C4.5 decision tree, design the algorithms and architectures for FPGA and multi-core platforms to achieve high throughput.

Note that the goal of this work is to achieve an online traffic classifier. The applications need to be detected and reported right after the system receives the first few packets of the incoming traffic flow. We introduce the details of the training and acceleration in Sections 4 and 5 respectively.

4 Machine Learning Based Training

Given a machine learning algorithm, the main challenge in building a highly accurate machine learning based traffic classifier is to choose an appropriate set of features that are effective in distinguishing the target applications. In this section, we discuss the training data set and the methodology for identifying the feature set.

As mentioned in Section 3, we build our traffic classifier based on the C4.5 decision tree algorithm [29]. The reason is that C4.5 based classifiers have demonstrated high classification accuracy for various target applications, test traces, and experiment setups [5], [30], [31].

We use WEKA [32], a well-known machine learning software, to train our decision trees. We feed WEKA with different feature sets to train C4.5 decision trees. WEKA outputs the decision trees along with the classification accuracy. The decision trees that achieve the highest overall accuracy are used in our implementations.

4.1 Dataset

Our dataset is built from two sources: a general traffic trace provided by a major network vendor, and a publicly available labeled traffic trace called Tstat [33]. Unlike many previous works targeting only one or two applications, our dataset consists of eight applications as listed in Table 2. These applications include Peer-to-peer (P2P) applications and Instance Messaging (IM) applications, which cannot be accurately classified using traditional port number or DPI based classification schemes. Each application in the dataset consists of 700 traffic flows. This dataset is used as both training set and testing set in our experiments.

4.2 Feature Selection

4.2.1 Candidate Features

Unlike many previous works which focus only on accuracy, we take the cost of computing into consideration. Specifically, we consider the following criteria when selecting the candidate features for the classifier.

- High Accuracy: The overall accuracy is the weighted average of *true positive rate* of all applications. True positive rate is the fraction of traffic flows correctly classified for an application.
- Early Classification: It is important to decide the application of the traffic flow by looking at only the first few initial packets. Therefore features characterizing a complete flow, such as duration, are not acceptable.
- Low Cost: The cost of calculating complex statistical features are usually high. This limits the usage of these features in online traffic classification. We constrain our selections to a set of low cost statistics of traffic flows (such as avg/min/max/variance of packet size).

In order to achieve high accuracy in classifying a traffic trace consisting of a broad range of applications, our work uses flow level features as opposed to packet level features used in many other works. We first identify eight candidate features which meet the above criteria. These features have been shown to be the most effective features for machine learning based traffic classification [1], [5], [6], [30], [34], [35], [36]. They are

- *Protocol*: the transport layer protocol associated with the flow.
- *Src./Dst. Port Number*: the source/destination port number associated with the flow.
- *Sizes of First N Packets*: sizes of the first N packets in the flow.
- *Max./Min. Packet Size*: maximum/minimum packet size among the first N packets in the flow.
- *Avg./Var. of Packet Size*: average/variance of sizes of the first N packets in the flow.

We group the above features into six sets as shown in Table 1. Each feature set in the table is a different combination of the above features. In this paper, we refer *Protocol*, *Src. Port Number* and *Dest. Port Number* as classic features. The classic features define end-to-end IP connections; hence many applications have their own designated port numbers (referred to as "well-known port numbers", e.g., 80 for HTTP). Because they are easy to extract from the network traffic and are accurate for applications using designated port numbers, we use classic features in all the feature sets.

We refer *Avg/Max/Min/Var Packet Size* as statistical features. In our experiments, those statistical features are computed over the first N packets in the flow. Previous works suggested to use the statistical features or the actual packet size [1], [5], [6], [30], [34], [35], [36]. Hence in this paper, we first show the experimental results on real life backbone router traces to decide whether we should use actual packet sizes, or their statistics in traffic classification, as well as the best value of N.

TABLE 1
Candidate Feature Sets

Feature set	Classic Features			Size of First N Pkts	Packet size statistics			
	Prtl.	Src. Port	Dst. Port		Avg. Pkt Size (byte)	Max. Pkt Size (byte)	Min. Pkt Size (byte)	Var. of Pkt Sizes
Set A	×	×	×					
Set B	×	×	×	×				
Set C	×	×	×		×	×	×	
Set D	×	×	×		×	×	×	×
Set E	×	×	×	×	×	×	×	
Set F	×	×	×	×	×	×	×	×

4.2.2 Methodology

Each feature set in Table 1 is evaluated using the same machine learning Algorithm (C4.5 Algorithm) and dataset. We apply the commonly used 10-fold cross validation technique [37] which breaks the dataset into 10 equally sized sets. In this technique, we perform 10 iterations; during each iteration, 9 sets are used as the training sets and 1 set is used as the testing set. The training sets are used to build the classifier and the testing set is used to evaluate the accuracy of the classifier. The overall accuracy of a feature set is computed by taking the average accuracy over 10 such iterations.

4.2.3 Discretization

Discretization is the process of converting continuous feature values into discrete feature values. In the discretization, the value space of a feature is separated into a finite number of ranges. A unique number is assigned to each range. Each input feature value is assigned the number associated with the range it falls into. Previous literature [38] has shown that the discretization can improve the accuracy of machine learning for traffic classification. We adopt Entropy-MDL [39], the most commonly used discretization algorithm, to separate the feature values space into discrete ranges. The sets of ranges for all the features are also used to discretize the input feature values in the online traffic classification (discussed in Section 4.3). All training and testing data are discretized.

4.2.4 Empirically Optimized Feature Set

We denote the feature set that achieves the highest overall accuracy as the empirically optimized feature set (EOFS). Note that, our methodology explores a limited design space using the available training data. Fig. 2 shows the overall accuracy achieved by each feature set. We observe that Feature Set C and Feature Set D achieve the highest accuracy among all feature sets. Feature Set C and Feature Set D differ only by the usage of packet size variance. Computing variance requires more resources; therefore, Feature Set C is favored over Feature Set D. Feature Set C achieves 97.92 percent overall accuracy, while Feature Set D achieves 98.02 percent. Not using variance only costs 0.1 percent of the overall accuracy.

From Fig. 2 we observe that maximum overall accuracy is achieved when using features from the first 4 packets in a flow. Having fewer or more than 4 packets in the feature set decreases the overall accuracy. This is because the P2P and IM flows use the first few packets for initial connection establishment. Thus the sizes and statistical properties of the first few packets best distinguish the applications. Too few packets cannot reveal this piece of information. Too many packets may involve the user data transmissions that are independent of the application. The user data transmission are usually "noise" to the classifiers, which reduces the classification accuracy.

Therefore, we choose Feature Set C computed over the first 4 packets in a traffic flow as our EOFS. All subsequent discussion is based on this EOFS and the C4.5 decision tree built using this EOFS.

4.2.5 Pitfalls of Classic Features

Our evaluation also demonstrates that the classic feature set (i.e., $Protocol$, $Src.PortNumber$, and $Dest.PortNumber$) is not sufficient in classifying a general traffic trace including P2P flows. Although the overall accuracy achieved by the classic feature set reaches 96.0 percent, it cannot distinguish various P2P applications.

TABLE 2
Applications in Our Traffic Traces

Application	Description
HTTP	Hypertext Transfer Protocol
MSN	MSN Messenger by Microsoft
P2PTV	P2P live streaming applications
QQ_IM	QQ Instant Messenger by Tencent
Skype	Skype voice calls and video calls
Skype_IM	Skype Instant Messenger
Thunder	P2P service by Thunder Networks
Yahoo_IM	Yahoo Instant Messenger

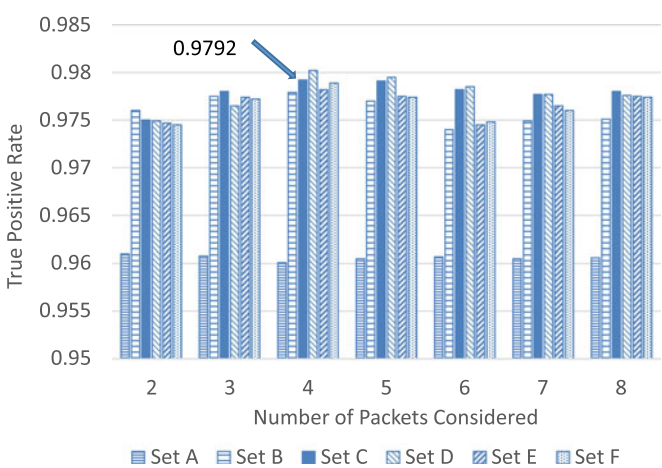


Fig. 2. Overall accuracy of each feature set.

TABLE 3
Confusion Matrix for Classic Feature Set

	HTTP	MSN	P2PTV	QQ_IM	Skype	Skype_IM	Thunder	Yahoo_IM
HTTP	**100.00**	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MSN	0.00	**99.95**	0.05	0.00	0.00	0.00	0.00	0.00
P2PTV	0.24	0.19	**98.00**	0.00	1.09	0.00	0.05	0.43
QQ_IM	0.39	0.00	0.00	**99.59**	0.00	0.00	0.03	0.00
Skype	0.00	0.00	1.15	0.00	**94.55**	0.05	4.25	0.00
Skype _IM	0.00	0.00	0.00	0.00	0.00	**100.00**	0.00	0.00
Thunder	0.00	0.00	0.12	0.00	8.00	0.02	**91.85**	0.00
Yahoo _IM	0.00	0.07	0.00	0.00	0.00	0.00	0.00	**99.93**

TABLE 4
Confusion Matrix for EOFS

	HTTP	MSN	P2PTV	QQ_IM	Skype	Skype_IM	Thunder	Yahoo_IM
HTTP	**99.90**	0.00	0.10	0.00	0.00	0.00	0.00	0.00
MSN	0.02	**99.93**	0.00	0.00	0.00	0.00	0.00	0.05
P2PTV	0.28	0.00	**99.19**	0.02	0.40	0.00	0.09	0.00
QQ_IM	0.28	0.00	0.00	**99.64**	0.00	0.03	0.05	0.00
Skype	0.00	0.00	0.34	0.05	**98.78**	0.07	0.76	0.00
Skype _IM	0.00	0.00	0.00	0.02	0.05	**99.93**	0.00	0.00
Thunder	0.00	0.00	0.02	0.07	1.60	0.02	**98.28**	0.00
Yahoo _IM	0.00	0.09	0.00	0.00	0.00	0.00	0.00	**99.91**

Tables 3 and 4 show the confusion matrices of classic feature set and our EOFS. The confusion matrix shows the fraction of traffic flows from an application that are classified as other applications. Each row index represents the application of incoming traffic; each column index represents the predicted application. For example, in Table 1, 0.24 percent of P2PTV traffic is classified as HTTP traffic.

It can be observed from Table 3 that the classic feature set cannot distinguish P2P traffic from each other. For example, 8 percent of the Thunder traffic has been falsely classified as Skype; 4.25 percent and 1.15 percent of the Skype traffic has been falsely classified as Thunder and P2PTV respectively. This causes the low overall accuracy of classic feature set shown in Fig. 2.

On the other hand, our EOFS can classify both P2P and non-P2P traffic with high accuracy. According to Table 4, in the worst case, only 0.76 percent of Skype traffic is falsely classified as Thunder traffic; only 1.60 percent of Thunder traffic is falsely classified as Skype traffic.

4.3 C4.5 Decision Tree Using EOFS

As mentioned in the beginning of Section 4.2, WEKA takes in the EOFS as the input and outputs the (binary) decision tree. Fig. 3 shows an example of the decision tree generated in our experiments and illustrates the process of online traffic classification. As shown in the figure, the online traffic classification has 2 phases: discretization and classification. In the discretization phase, the raw feature values are discretized based on the sets of ranges generated in the training process (discussed in Section 4.2.3); in the classification phase, the application is determined based on the discretized values. Let $F[i]$ denote the raw feature value of feature type i. Using our proposed EOFS, the input traffic flow can be represented by a set of M feature values $F[i]$, where $i = 0, 1, \ldots, M-1$. Let $\{D[i] : 0 \leq i < M\}$ denote output for $F[i]$ after the discretization phase. In the discretization phase, the features are assigned the discrete values associated with the ranges they fall into. In this example, we suppose $D[0]$ and $D[1]$ have 3 and 2 possible values respectively. $\{D[i] : 0 \leq i < M\}$ are used as inputs to the classification phase. The classification phase involves a binary search in the decision tree. Let T_c denote the binary decision tree. Each non-leaf node of T_c corresponds to one possible discretized value of one feature. Each leaf node corresponds to an application. During the decision tree search, when a non-leaf node is reached, the input value for the node's feature are compared with the node's value. The outcome of each comparison can be either "*Yes*", denoted as a dashed green edge in Fig. 3, or "*No*", denoted as a solid red edge in Fig. 3. The classification terminates when a leaf node is reached, where the traffic type can be reported. In this example, we examine $M = 2$ features ($F[0]$ and $F[1]$) collected from the input traffic flow. The output is QQ_IM.

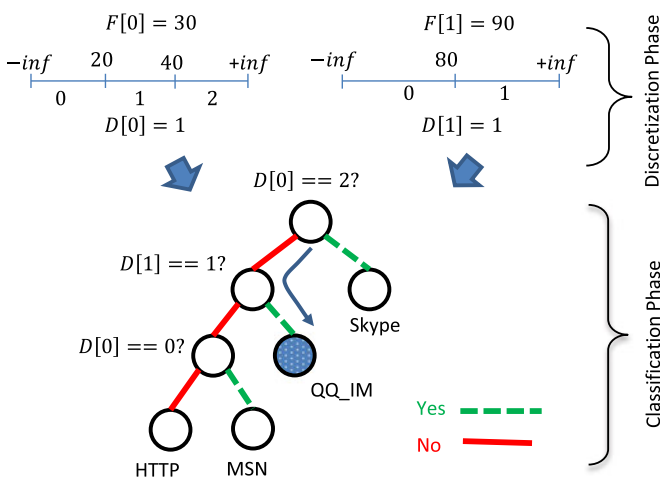


Fig. 3. An example of C4.5 decision tree using EOFS & the process of online traffic classification.

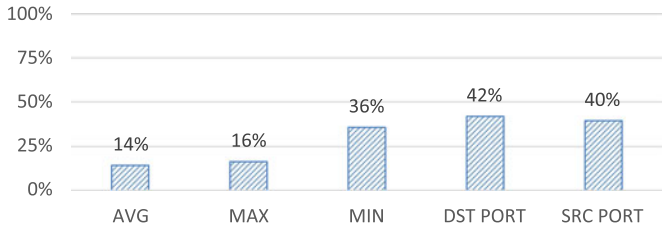


Fig. 4. Percentage of the discretized feature values used during classification.

5 Acceleration

Based on the decision trees trained with EOFS, in this section, we present two algorithms for online traffic classification. In Section 5.1, we present an algorithm that combines the discretization and the C4.5 decision tree together. In Section 5.2, we present a divide and conquer algorithm by first translating the decision tree into multiple *range trees*. Then we develop their corresponding acceleration architectures on FPGA and multicore platforms in Section 5.4 and Section 5.5.

5.1 Optimized Decision Tree (ODT)

In Fig. 3, we observe that:

- The total number of possible discretized values is much larger than the total number of non-leaf nodes in the decision tree.
- Different non-leaf node can contain the same discretized value of the same feature.

The percentage of used discretized feature values in the EOFS over all the decision trees are shown in Fig. 4. As shown in these figures, the decision trees contain much smaller number of discretized value compared with total number of discretized value in the discretizer. Therefore, the classification algorithm in Fig. 3 stores redundant discretized values in the discretizer.

To address this issue, we combine the discretizers and T_c. Instead of generating $D[i]$s and performing a decision tree search, we substitute the discretized values in the non-leaf nodes by the range boundaries associated with the discretized values. In this way, at each non-leaf node we perform comparisons with the 2 range boundaries to decide whether to take the left or right branch to the next tree node. The key idea is to eliminate the entire discretization phase by replacing the single comparison in each non-leaf node in T_c with at most 2 comparisons. In this way, all the redundant

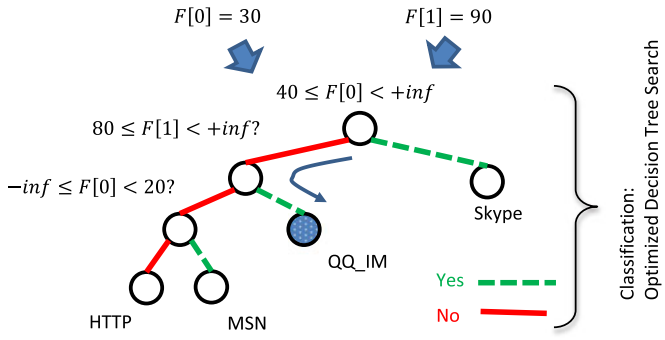


Fig. 5. An optimized decision tree T_s integrated by two discretization trees ($T[0]$, $T[1]$) and one decision tree (T_c).

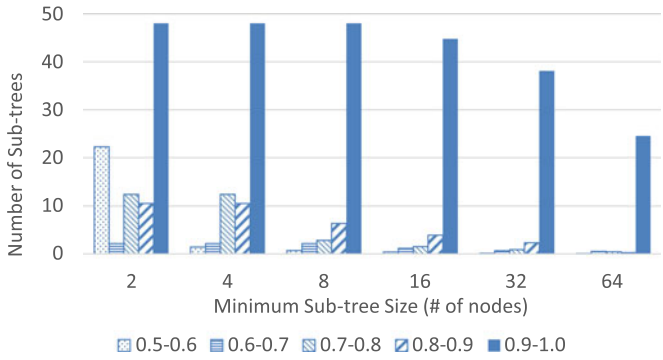


Fig. 6. Number of sub-trees in each balance factor ranges.

discretized values and their associated ranges are removed from the trees. We use T_s to denote this optimized decision tree. The example in Fig. 5 combines the discretizer and the decision tree in Fig. 3. Comparing the 2 figures, we can see the advantage of T_s. Since the shape of the decision trees in these two figures are the same, to arrive at QQ_IM in the decision tree search, T_s needs only 1 additional comparison at each non-leaf node compared with T_c. In total, T_s performs 2 additional comparisons than T_c. However T_c needs much more comparisons to discretize the input feature values. Therefore T_s not only reduces the memory usage but also requires a smaller number of comparisons. Since the algorithm to translate T_c into T_s is not complicated, we ignore further discussion of this algorithm in this paper. Throughout the rest of the paper, we use ODT to denote this approach.

5.2 Divide and Conquer (DQ)

Using the algorithm in Section 5.1, the worst case latency for online traffic classification is determined by the number of decision tree levels. Therefore for a given number of leaf nodes, imbalanced trees have longer latency. Fig. 6 shows the *balance factor* of the sub-trees in our decision tree trained using EOFS. The *balance factor* is defined as the proportion of the nodes that reside in the more populated branch of the sub-tree root. For example, in Fig. 5, the root has a balance factor of $5/6 = 0.83$, the lowest non-leaf node has a balance factor of $1/2 = 0.5$ (i.e., it is a balanced sub-tree). As shown in Fig. 6, the size of the balanced sub-trees are very small and almost all sub-trees that have considerable sizes have over 90 percent of their nodes in one of its branches. Given that the sizes of the decision tree are around 200 nodes, this means the decision tree trained using EOFS is highly imbalanced. As a result, the online traffic classification can have long latency.

Our goal is to find a parallel data structure that leverages balanced binary trees. Essentially, we need to determine which ranges the M input feature values belong to. Note that the ranges we mention in this section is the ranges created by the boundaries used in the T_s. The search process in T_s ends at a leaf node where an application is found. At this point, all the ranges for the input features have been determined. This process can also be done by first separately determining the range for each feature, then merging the searching results. Since we have a known set of possible ranges from the training process, we can construct complete binary search trees to determine the ranges for the features. We denote these trees as $R[i]$, where $i = 0, 1, \ldots, M-1$. Throughout the rest of the paper, we use *range tree* to refer to these trees.

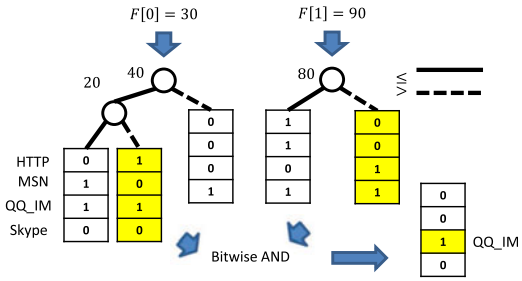


Fig. 7. Divide and conquer: using multiple range trees.

In Fig. 7, we show an example of our divide and conquer approach. The two range trees are constructed from T_s in Fig. 5. Each non-leaf node corresponds to a boundary of the ranges. The search process proceeds based on the result of the comparison between the input and the boundaries stored in each non-leaf node. Each leaf node of a range tree stores a *Bit Vector* (BV). Each bit in a BV corresponds to an application, and indicates whether the input matches the corresponding application. In this example, by comparing the discrete value $F[0] = 30$ with two range boundaries, a BV (highlighted in the figure) is selected as a partial result for this feature. Similarly we also collect a BV for $F[1] = 90$. The final classification result can be reported by performing a bitwise AND operation on the two selected BVs. In this example, the final BV after the bitwise AND operation indicates the incoming traffic type is "QQ_IM". Throughout the rest of the paper, we use DQ to denote this divide and conquer approach.

We show the algorithm to construct M range trees in Algorithm 1. In this algorithm:

- Step 1: The boundary pool is a set of all the range boundaries.
- Step 2: This is done by going through all the branches of T_s.

5.3 Comparison Between ODT and DQ

Suppose we use complete binary range trees in DQ, and all the range trees are searched in parallel, the time complexity of DQ is $\mathcal{O}(\max_i(\log n_i) + M \cdot K)$, where n_i is the number of boundaries in the ith range tree. For our problem M and K are both small, the time to bitwise-AND the M K-bit BVs is negligible. Fig. 8 shows the number of levels of each decision tree trained using EOFS, and the total number of steps to search each of the 42 range trees (generated using real life traffic trace). As shown in the figure, the number of steps to search the range trees is much smaller than the number of tree levels. And it does not vary dramatically even as the number of decision tree levels doubles.

The complexity of ODT is linearly proportional to the number of levels of the decision tree. Therefore, considering the imbalance of the decision trees, when M and K are small, DQ has great advantage over ODT. However as demonstrated in Section 6, when M, K, and n vary, there are still scenarios in which ODT outperforms DQ.

5.4 Acceleration on FPGA Platform

On FPGA, we propose pipelined architectures for both ODT and DQ. Each clock cycle, the architecture consumes one input and generates one output.

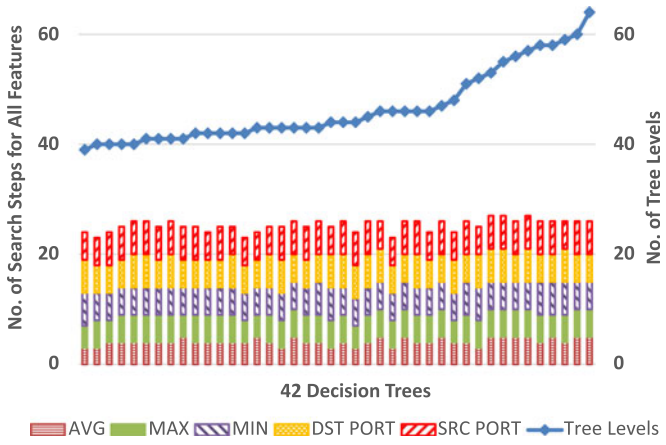


Fig. 8. Number of steps to search for all the features in DQ.

Algorithm 1. Constructing Range Trees

Input: M discretization trees and a T_c having N leaves.

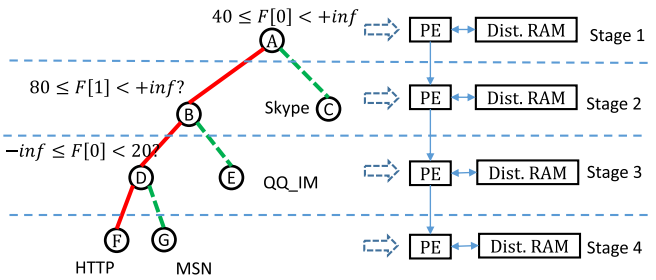
Output: Range trees $R[i]$, $i = 0, 1, \ldots, M-1$; $R[i]$ is a complete binary tree where each non-leaf node stores a range boundary, and each leaf node stores a BV denoted as $B[i,j] = \overline{b_0^{i,j} b_1^{i,j} \ldots b_{K-1}^{i,j}}$, where $j = 0, 1, \ldots, n_i - 1$. Let n_i denote the number of leaf nodes in $R[i]$; K is the number of applications.

Initialization: Construct T_s according to Section 5.1. Set all the BVs to all-zero vector.

1: **for** $i = 0 : M-1$ **do**
2: boundary pool $\leftarrow \emptyset$
3: **for** each non-leaf node x' of T_s **do**
4: **if** x' checks $F[i]$ **then**
5: Add the two range boundaries stored in x' into the boundary pool
6: **end if**
7: **end for**
8: Construct a complete binary tree $R[i]$ based on the boundaries recorded in the boundary pool
9: **for** each leaf node of $R[i]$ **do**
10: **for** $k = 0 : K-1$ **do**
11: **if** application k appears in the corresponding interval in T_s **then**
12: $b_k^{i,j} \leftarrow 1$
13: **else**
14: $b_k^{i,j} \leftarrow 0$
15: **end if**
16: **end for**
17: **end for**
18: **end for**

5.4.1 Optimized Decision Tree

Fig. 9 shows our acceleration architecture on FPGA for ODT. To achieve high throughput, we use a deeply pipelined architecture. Each level is mapped to a pipeline stage. The decision tree nodes are stored as words in the Distributed RAM (denoted as Dist. RAM). The memory layout for both the leaf and non-leaf nodes are also shown in Fig. 9. In each stage, the processing element (PE) retrieves the tree node from the Dist. RAM, performs the comparisons, and determines the node to access in the next stage. The input is forwarded to the next stage along with the address of the target child node.



Node A (non-leaf)

Is_leaf	Feature type	Lower bound	Upper bound	Address of the left child in the next stage	Address of the right child in the next stage
0	000	40	$-inf$	Address of Node B	Address of Node C

Node C (leaf)

Is_leaf	Class_ID
1	11 (Skype)

Fig. 9. Mapping ODT to FPGA.

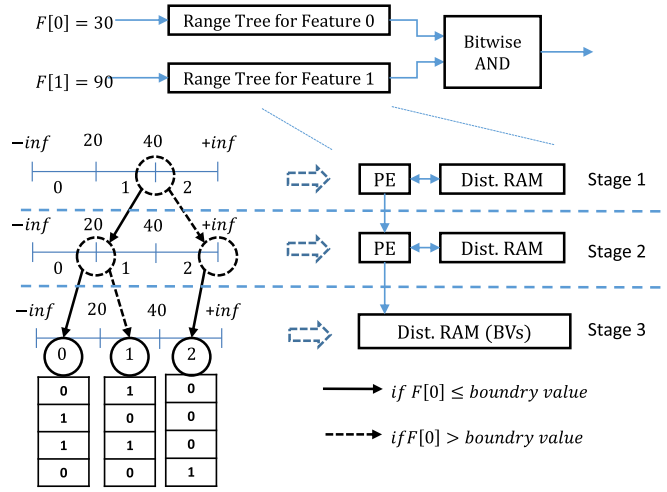


Fig. 10. Mapping DQ to FPGA.

5.4.2 Divide and Conquer

Fig. 10 shows our acceleration architecture on FPGA for DQ. Each range tree is mapped to a separate pipeline; the outputs of the range trees are merged in a bitwise AND module to generate the final output. As discussed in Section 5.2, the range trees are organized as complete binary search trees. The mid-point of all the range boundaries is used as the root. All the boundaries smaller (larger) than the mid-point are mapped to the left (right) sub-trees. The left and right sub-trees are recursively defined similarly. Each level of the range tree is mapped to a pipeline stage. The nodes at each level are stored in Dist. RAM. The PE compares the input with the boundary values and generates the address to access in the next stage. The leaf level of the range tree is mapped onto the last stage of the pipeline.

5.5 Acceleration on Multicore Platform

For ODT, we use the classic implementation of trees: all the nodes are accessed through pointers. The decision tree is searched level by level. In each level two comparisons are performed to decide which branch to take; the target nodes are accessed through the pointers.

For DQ, the range trees are implemented as array based binary search trees. The BVs are implemented using integer type and maintained in a separate array from the range trees; they are accessed after the range trees have been searched.

The reasons we used different tree implementations for DQ and ODT are:

- T_s has the same shape as T_c (very imbalanced); thus, pointer based implementation saves memory.
- Range trees are complete binary search trees, using arrays eliminates the use of pointers and speeds up the search process

6 Experimental Results

In this Section we show the performance of our algorithms and architectures for the decision trees trained using EOFS, and compare the performance of our classifier with other techniques. We also compare the performance of ODT and DQ. We generate 42 decision trees using real life network traffic traces. All the statistics shown in this section are computed over these 42 trees. In the experiments, the algorithms and architectures are configured based on these statistics unless otherwise stated.

6.1 Experimental Setup

6.1.1 Platforms

We evaluate our accelerator on both multicore and FPGA platforms. For the multicore platforms, we use a 2× AMD Opteron 6278 platform [16] and a 2× Intel Xeon E5-2470 platform [17]. The dual-socket AMD platform has 16 physical cores, each running at 2.4 GHz. Each core is integrated with a 16 KB L1 data cache and a 2 MB L2 cache. A 6 MB L3 cache is shared among all 16 cores. The processor has access to 64 GB DDR3-1600 main memory. The dual-socket Intel platform also has 16 cores, each running at 2.3 GHz. Each core has a 32 KB L1 data cache and a 256 KB L2 cache. All 16 cores share a 20 MB L3 cache. The processor has access to 48 GB DDR3-1600 main memory. We implemented our classifiers in C++ on openSUSE 12.2 and boosted up the performance using Pthread. For both ODT and DQ, each thread carries a traffic classifier. There is no interaction among various threads. The algorithms are implemented as normal user space programs. No special tools or techniques are used to bypass the operating system. The test flows are fed to the classifier from memory and the classification results are written back into the memory.

For the FPGA platform we use a state-of-the-art Xilinx Virtex Ultrascale XCVU440-FLGA2892-3-E-ES2 FPGA. All the results are post place-and-route results based on Xilinx Vivado 2015.4. The implementation is in RTL; no high level synthesis tool is used. The target device provides abundant logic and storage resources, so we are able to map multiple classifiers on a single FPGA device.

6.1.2 Performance Metrics

We use throughput as the performance metric. Throughput is defined as the number of classifications completed per unit time. We use Million Classifications Per Second (MCPS) as the unit for throughput.

On the multicore platforms, the throughput is computed by dividing the number of test flows with the time spent in

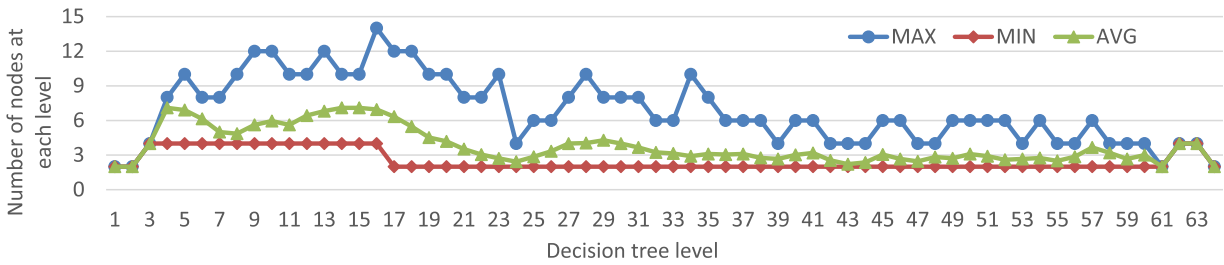


Fig. 11. Number of nodes at each decision tree level.

classifying these flows. The time we recorded includes thread control and co-ordination overhead.

On the FPGA platform, in each clock cycle the classifier can take in one input and output one classification result. Therefore, the clock rate is equal to the throughput of the classifier numerically. We also report the FPGA resource usage. The resource usage is an important indication of the hardware complexity of our design. The logic resources on Xilinx FPGA devices are organized using slices containing LUTs and registers. All the data structures are maintained using on-chip Dist. RAM. Since Dist. RAM is built using LUTs we further break the LUT usage to LUT as logic and LUT as Dist. RAM. We report the usage in percentage. The device we use has 2.5 M slice LUTs, and 5 M slice registers. Among the 2.5 M slice LUTs, 0.46 M can be mapped to Dist. RAM.

6.2 Performance for Real Decision Trees

6.2.1 FPGA Platform

We configure the architecture for ODT based on the statistics of the decision trees. Fig. 11 shows the statistics of the number of nodes in each decision tree level. The maximum number of nodes per level is 14, so we assign Dist. RAM for 16 nodes in each pipeline stage to make sure that all the decision trees can fit. The number of features and the applications are fixed to 6 and 8, respectively. So we use 3 bits for both of them. The number of tree levels determines the number of pipeline stages. From Fig. 8, we can observe that the number of tree levels ranges from 39 to 64. In general, due to the limited resources per unit area on the chip, higher resource consumption forces components to be mapped farther away from each other resulting in longer routing latency. Therefore, given an architecture on FPGA, the more resource a certain configuration uses, the lower the maximum clock rate it can achieve. Since trees with more levels consume more resources, they usually have lower clock rate. Therefore we test the cases of 39 levels and 64 levels to show the throughput for our architecture on FPGA, respectively.

For DQ, since there are 8 applications, we use 8 bits for each BV. We have 6 pipelines, each for one feature. These pipelines are configured to be able to accommodate the feature with the most number of boundaries (after processing the decision tree using Algorithm 1). For our 42 decision trees, the number of boundaries to support ranges from 31 to 49. Similar to ODT, more resource usage leads to lower clock rate. Therefore we test the cases for 31 and 49 boundaries, to show the throughput of our architecture on FPGA.

Fig. 12 shows the result using a single pipeline. DQ achieves much higher throughput than ODT with a much lower resource consumption.

We also implement multiple pipelines on one FPGA device to boost up the throughput. Fig. 13 shows the results for 2, 8, and 16 pipelines. Using 16 pipelines the throughput

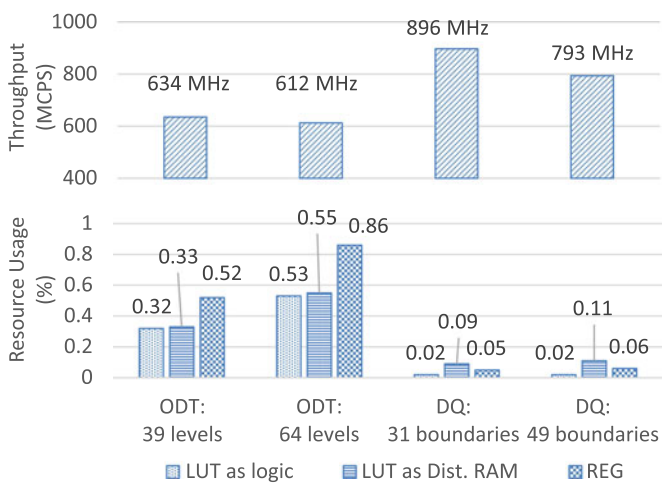


Fig. 12. Performance on FPGA: single pipeline.

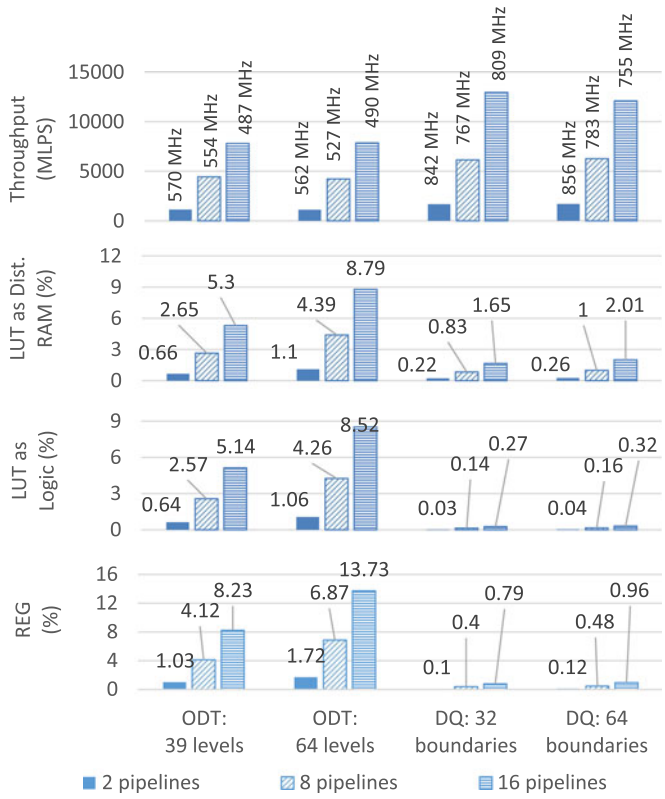


Fig. 13. Performance on FPGA: multiple pipelines.

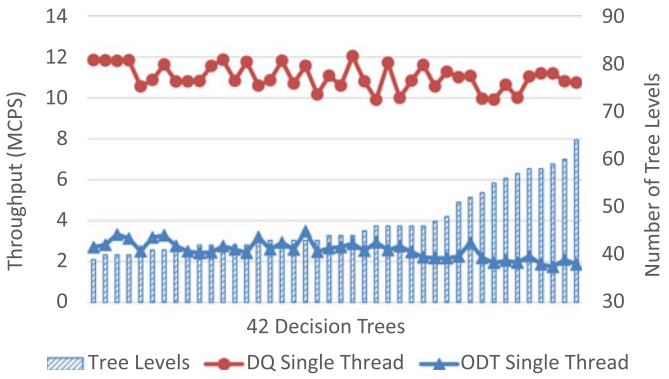


Fig. 14. Throughput on the AMD platform: real decision trees, tested with real network flows, single classifier thread.

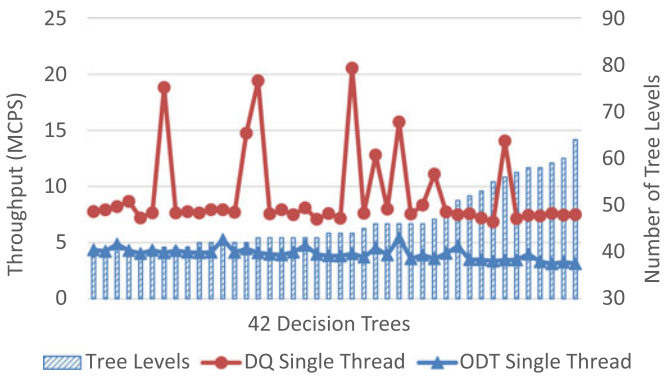


Fig. 15. Throughput on Intel platform: real decision trees, tested with real network flows, single classifier thread.

can be boosted to over 10,000 MCPS. Since the area complexity does not change when we use multiple pipelines, the resource usage is the product of the results in Fig. 12 and the number of pipelines.

6.2.2 Multicore Platform

We first test the performance on multicore platform using real life traffic flows captured from ISP routers. They best reflect the features of real life network traffic, such as the distribution of applications and packet sizes. Since we do not have enough real life flows to keep multiple threads running at full speed concurrently, we use single classifier thread in the experiments for real life network traffic. As shown in Fig. 14, on the AMD platform, DQ achieves 12 MCPS, while ODT only achieves around 4 MCPS. For all the decision trees, the throughput of DQ is 3-4 times higher than ODT. Similar results can be observed on the Intel platform as well, as shown in Fig. 15.

To test the highest throughput our algorithms can achieve on our platforms, we accelerate them using multiple concurrent threads. We generate a large number of synthetic test flows to feed the classifier threads to keep them working concurrently. Both of our platforms have 16 cores; each core can support 2 threads to work concurrently. So we allocate 32 threads on both platforms. The results are shown in Figs. 16 and 17. On the AMD platform, DQ achieves 230+ MCPS and ODT achieves 75+ MCPS. On the Intel platform, the performance fluctuation of both algorithms are greater than on the AMD platform. DQ achieves 130-350 MCPS for various decision trees. ODT achieves 50-190 MCPS for various decision

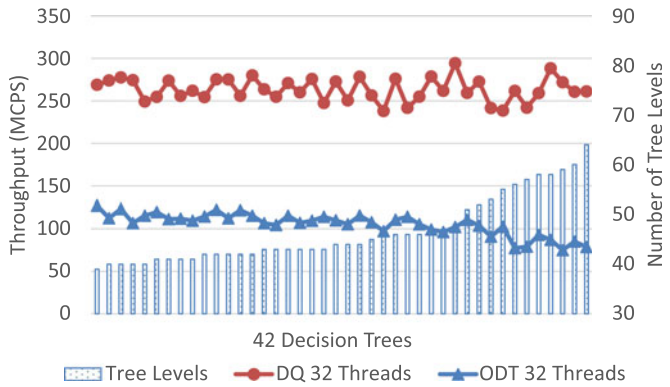


Fig. 16. Throughput on the AMD platform: real decision trees, tested with synthetic flows, 32 classifier threads.

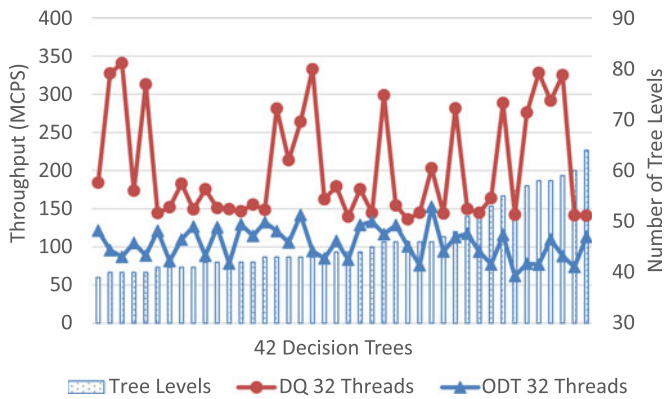


Fig. 17. Throughput on the Intel platform: real decision trees, tested with synthetic flows, 32 classifier threads.

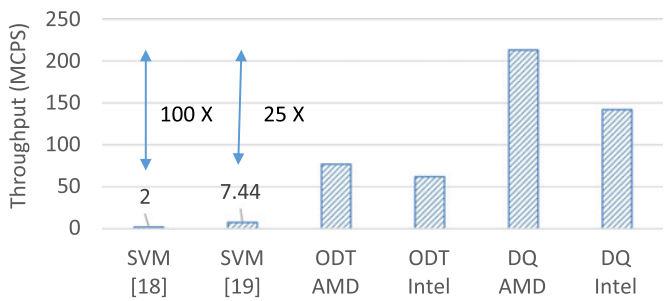


Fig. 18. Throughput comparison of our proposed approaches with other techniques in the research literature.

trees. For most of the decision trees using EOFS, DQ achieves much higher throughput than ODT on both platforms. For some decision trees the performance advantage of DQ over ODT is lower on the Intel platform.

We have noticed the performance fluctuations in Figs. 15 and 17; this is because the performance of our user-space programs is adversely affected by random OS events.

6.2.3 Comparison with Other Techniques

Fig. 18 compares the throughput of our approaches with state-of-the-art SVM based approaches [26], [27]. The throughput values of our approaches in Fig. 18 are the minimum values in Figs. 16 and 17. In [26], the performance was achieved on a dual Xeon PC with 24 cores at 2.6 GHz with 48 GB of RAM. In [27] the performance was achieved on a a dual Xeon PC with 12 cores at 2.6 GHz with 24 GB of RAM.

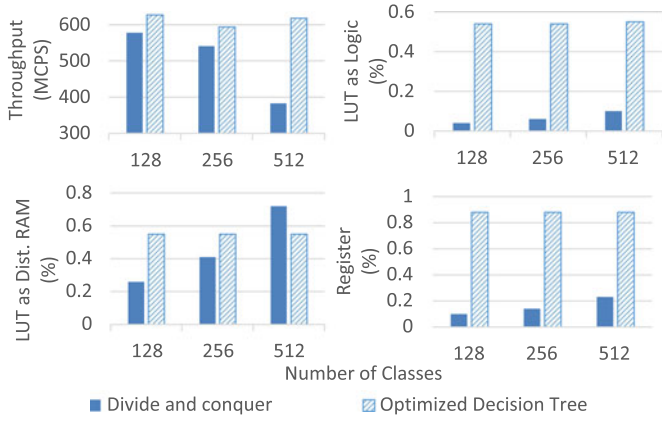


Fig. 19. Comparison of the two algorithms on FPGA: large number of applications.

Both of our approaches achieve great improvement compared with the SVM based approaches.

6.3 Comparison Between ODT and DQ

As shown in Section 6.2, for all the decision trees using EOFS, DQ achieves higher throughput on both platforms, and consumes less resources on the FPGA than ODT. In this section, we vary the configuration used in Section 6.2 to study how various configurations impact our algorithms and architectures.

6.3.1 FPGA Platform

On FPGA, the various parameters discussed in this section (6.3.1) are set to be able to accommodate trees of up to 64 levels with up to 16 nodes on each level. This results in a maximum number of 1024 nodes in total[1].

The number of applications, K. In a binary decision tree of 1023 nodes, there are 512 leaf nodes (no matter what shape the tree is because each non-leaf node must have two children), so there can be up to 512 applications. Fig. 19 shows the throughput and resource usage of ODT and DQ for up to 512 applications. As shown in the figure, the throughput of DQ decreases dramatically; the resource usage of DQ increases dramatically. On the contrary the throughput and resource usage of ODT varies little. The reason is as follows. In ODT, the K applications can be represented using $\log_2 K$ bits. In DQ, K applications need K-bit BVs. So when K gets large, DQ needs much more storage for the applications than ODT. Also, in DQ, for each additional bit, we need additional LUT logic to perform the bitwise AND operations. Therefore, when the number of applications increases, the resource consumption of DQ grows much faster than ODT. Thus the throughput of DQ decreases much faster than ODT as the number of applications increases.

The number of features, M, and the number of boundaries per feature n_i $(i = 0, 1, 2 \ldots M-1)$: In our experiments, the binary decision tree has 1024 nodes. There are 512 non-leaf nodes. Since each non-leaf node has 2 boundaries, when all boundaries are different, the total number of boundaries $\sum_{i=0}^{M-1} n_i = 1024$. Fig. 20 shows the experimental results for various M. As can be observed when M grows, the resource consumption grows rapidly, and the throughput first increases and then decreases. The post-place-and-route reports show that when $M = 2$, the critical path resides in the routing to the Dist. RAM, while for all the other cases the critical path is in the PE. This explains why the throughput first increases then decreases. When $M = 2$, the Dist. RAM blocks for the BVs are large. The long access latency to the large Dist. RAM block is the bottleneck of the architecture. When M increases, the size of the Dist. RAM blocks shrinks and the increasing resource consumption forces the components in the PEs to be mapped farther apart, which becomes the new bottleneck. The increasing resource usage is due to the increasing total number of pipelines stages for all the features. According to the discussion in Section 5.4, the total number of pipeline stages is $\sum_{i=0}^{M-1}(\lceil \log_2 n_i \rceil + 1)$. When M increases, the total number of pipeline stages increases. For example, we can split n_0 into two parts n_0' and n_0'' $(n_0', n_0'' > 0)$, while preserving the number of boundaries for the other features. In this case the number of features increases by 1. The original number of pipeline stages for feature 0 are $\log_2\lceil (n_0' + n_0'') \rceil + 1$, this number is no less than the number after the split, $\log_2\lceil n_0' \rceil + \log_2\lceil n_0'' \rceil + 2$.

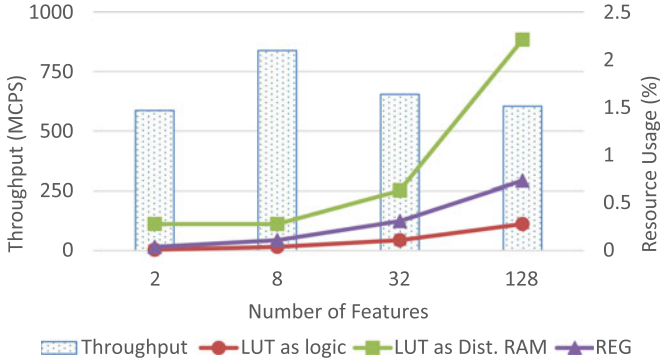


Fig. 20. Throughput and resource usage of DQ for various number of features.

The performance of ODT for decision trees of up to 64 levels with up to 16 nodes on each level is shown in Fig. 20 (1024 nodes in total, regardless of M and n_i). We can compare it with the results of DQ in Fig. 12 to compare the performance of ODT and DQ for decision trees of various M. When $M \geq 32$, DQ does not have any advantage over ODT. When $M \leq 8$, DQ either achieves higher throughput with similar resource consumption or achieves similar throughput with lower resource consumption than ODT.

6.3.2 Multicore Platform

On the two multicore platforms, we vary K, M, and n based on the real decision trees. We also discuss the scenarios where the decision tree searches in ODT finishes in few levels.

The number of applications, K. Fig. 21 shows the average throughput of both approaches for the 42 real decision trees with various number of applications. Since the decision trees have no more than 256 nodes, they cannot have more than 128 leaf nodes. So we use $K \leq 128$. As shown in the figure, for 16, 32, 64 applications, when we can use built-in data types such as short integer type, integer type, and long

1. The actual number of nodes is less than 1024, since the first 4 levels from the root level have less than 16 nodes each. Without losing generality, we round it up to 1024 nodes to make the discussion easy to follow. The numbers of leaf and non-leaf nodes are also rounded up

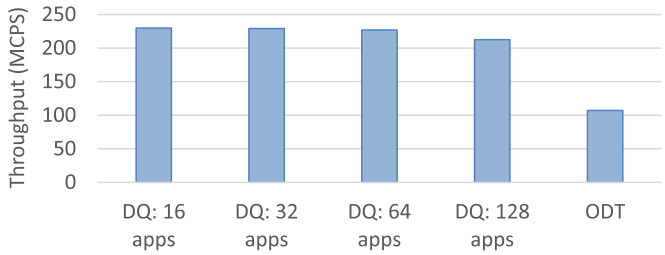


Fig. 21. Average throughput on the AMD platform: real decision trees with various synthetic number of applications, synthetic test flows.

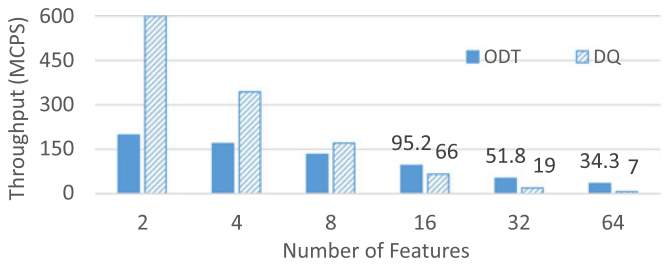


Fig. 22. Average throughput on the AMD platform: real decision trees with various synthetic number of features, synthetic test flows.

integer type to store the BVs, the throughput of DQ is steady and much higher than ODT. In these cases, merging the BVs only needs M bitwise AND operations. Once the number of applications exceeds the capacity of the built-in data types, we need to use arrays of built-in date types to store the BVs and iterations to merge the BVs. A larger number of applications requires more storage to store the BVs, and more iterations to merge the BVs; this leads to a lower throughput. Since $K \leq 128$ for our decision trees, we can use two 64-bit long integers to store each BV. Thus, we only need up to $2M$ bitwise AND operations and the throughput is not dramatically affected. However, when the decision tree supports a very large number of applications, ODT can have a higher throughput than DQ.

The number of features, M, and the number of boundaries per feature n_i ($i = 0, 1, 2 \ldots M-1$): The analysis on M in Section 6.3.1 is also valid for multicore platform. On the multicore platform, $\sum_{i=0}^{M-1}(\lceil \log_2 n_i \rceil + 1)$ represents the number of steps to search all the range trees. Fig. 22 shows the average throughput over the decision trees of various synthetic M. These decision trees are generated by modifying the features and boundaries used in the decision trees generated in the classifier training, while preserving their shapes. As shown in the figure, as M increases the throughput of DQ decreases. The results in the fig. complies with our analysis in Section 6.3.1. Note that when M increases the amount of memory to store the test flows also increases, this is why the throughput of ODT also decreases as M increases.

Search in ODT finishes in few levels. The levels to search in ODT can be reduced in the following scenarios: 1. given a fixed number of nodes, the tree is balanced; 2. most of the test flows target the leaf nodes close to the root. Fig. 23 shows the throughput comparison between the two approaches for synthetic trees of various number of nodes and various average balance factors over all the non-leaf nodes. As can be observed, the throughput of ODT decreases dramatically as the trees become more imbalanced and as the number of nodes increases. The

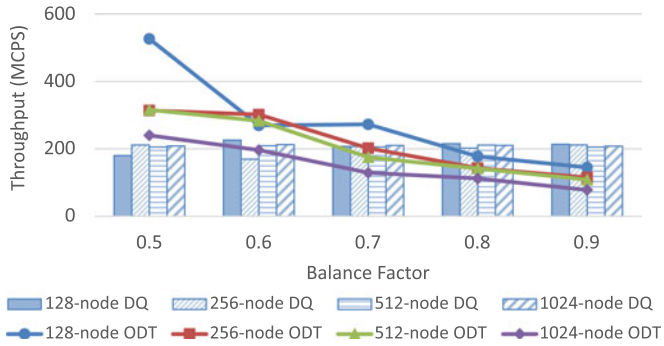


Fig. 23. Average throughput on the AMD platform: synthetic decision trees of various balance factors and various number of nodes, synthetic test flows.

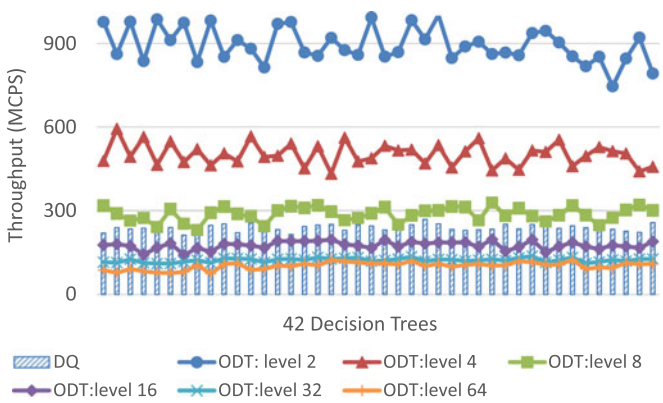


Fig. 24. Average throughput on the AMD platform: real decision trees, synthetic test flows whose search ends within various levels from the root.

throughput of DQ varies little as the balance factor and the number of nodes changes. Thus, ODT is good for the more balanced decision trees with small number of nodes, while the DQ approach is good for the imbalanced decision trees with large number of nodes. Fig. 24 shows the result for the real decision trees tested with flows targeting the leaf nodes within a certain number of levels from the root. We can observe that for all the decision trees trained using EOFS, when all the test flows target the leaf nodes within 8 levels from the root, ODT has higher throughput than DQ. Otherwise DQ has higher throughput.

7 Conclusion

In this paper, we first selected a set of traffic features for accurate classification; the empirically optimized feature set consists of transport layer protocol, source port number, destination port number, average packet size, maximum packet size, and minimum packet size. Using this feature set, we sustained a high classification accuracy of over 97 percent. Then we proposed two algorithms by optimizing decision tree, or by a divide and conquer technique. We designed architectures for these approaches on FPGA to achieve high throughput. The DQ and ODT achieved 10000+ and 8000+ MCPS throughput respectively on a state-of-the-art FPGA. We also presented implementations on multicore platforms. The DQ and ODT achieved 230+ and 80 + MCPS throughput on state-of-the-art multicore processors. Through extensive experiments and analysis we demonstrated that, for the decision trees trained using EOFS, DQ

achieves higher throughput with lower resource consumption than ODT. But in the scenarios where the number of applications and number of features are large, ODT achieves higher throughput on both multicore and FPGAplatforms.

It will be interesting to explore heterogeneous platforms for traffic classification. For example, the Zynq-7000 All Programmable System-on-Chip (APSoC) [40] is an attractive platform; on such a platform both software and hardware engines can be deployed.

Acknowledgments

This work is supported by the U.S. National Science Foundation (NSF) under grants No. CCF-1320211 and No. ACI-1339756. Equipment grant from Xilinx is gratefully acknowledged.

References

[1] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 2, pp. 23–26, 2006.

[2] Y. Luo, K. Xiang, and S. Li, "Acceleration of decision tree searching for IP traffic classification," in *Proc. 4th ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, 2008, pp. 40–49.

[3] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, "Deep packet inspection using parallel bloom filters," *IEEE Micro*, vol. 24, no. 1, pp. 52–61, Jan./Feb. 2004.

[4] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: Multilevel traffic classification in the dark," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 229–240, 2005.

[5] R. Alshammari and A. N. Zincir-Heywood, "Machine learning based encrypted traffic classification: Identifying SSH and skype," in *Proc. IEEE Symp. Computational Intell. Sec. Defense Appl.*, 2009, pp. 289–296.

[6] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: Myths, caveats, and the best practices," in *Proc. ACM CoNEXT Conf.*, 2008, pp. 11:1–11:12.

[7] W. Jiang and M. Gokhale, "Realtime classification of multimedia traffic using FPGA," in *Proc. Int. Conf. Field Programmable Logic Appl.*, 2010, pp. 56–63.

[8] R. Garcia, I. Algredo-Badillo, M. Morales-Sandoval, C. Feregrino-Uribe, and R. Cumplido, "A compact FPGA-based processor for the secure hash algorithm SHA-256," *Comput. Elect. Eng.*, vol. 40, no. 1, pp. 194–202, 2014.

[9] B. Humphries, H. Zhang, J. Sheng, R. Landaverde, and M. C. Herbordt, "3D FFTs on a single FPGA," in *Proc. IEEE 22nd Auuu. Int. Symp. Field-Programmable Custom Comput. Mach.*, May 2014, pp. 68–71.

[10] B. Yang, J. Fong, W. Jiang, Y. Xue, and J. Li, "Practical multituple packet classification using dynamic discrete bit selection," *IEEE Trans. Comput.*, vol. 63, no. 2, pp. 424–434, Feb. 2014.

[11] J. McGlone, R. Woods, A. Marshall, and M. Blott, "Design of a flexible high-speed FPGA-based flow monitor for next generation networks," in *Proc. Int. Conf. Embedded Comput. Syst.*, Jul. 2010, pp. 37–44.

[12] M. Attig and G. Brebner, "400 Gb/s programmable packet parsing on a single FPGA," in *Proc. ACM/IEEE 7th Symp. Archit. Netw. Commun. Syst.*, 2011, pp. 12–23.

[13] D. Tammaro, E. A. Doumith, S. A. Zahr, J.-P. Smets, and M. Gagnaire, "Dynamic resource allocation in cloud environment under time-variant job requests," in *Proc. IEEE 3rd Int. Conf. Cloud Comput. Technol. Sci.*, 2011, vol. 0, pp. 592–598.

[14] A. Manzalini, R. Minerva, F. Callegati, W. Cerroni, and A. Campi, "Clouds of virtual machines in edge networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 63–70, Jul. 2013.

[15] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *Proc. INFOCOM*, 2013, pp. 545–549.

[16] AMD Opteron 6200 Series Processor. [Online.] Available: http://www.amd.com/us/products/server/processors/6000-series-platform/6 200/Pages/6200-series-processors.aspx.

[17] Intel Xeon Processor E5–2470. [Online.] Available: http://ark.intel.com/products/64623/Intel-Xeon-Processor-E5–2470-20M-Ca che-2_30-GHz-8_00-GTs-Intel-QPI?wapkw=e5+2470.

[18] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 50–60, 2005.

[19] Z. Lin, C. Lo, and P. Chow, "K-means Implementation on FPGA for high-dimensional data using triangle inequality," in *Proc. 22nd Int. Conf. Field Programmable Logic Appl.*, 2012, pp. 437–442.

[20] M. Papadonikolakis and C. Bouganis, "Novel cascade FPGA accelerator for support vector machines classification," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 23, no. 7, pp. 1040–1052, Jul. 2012.

[21] A. Este, F. Gringoli, and L. Salgarelli, "Support vector machines for TCP traffic classification," *Comput. Netw.*, vol. 53, no. 14, pp. 2476–2490, 2009.

[22] T. Bakhshi and B. Ghita, "On internet traffic classification: A two-phased machine learning approach," *J. Comput. Netw. Commun.*, vol. 2016, p. 21, 2016, Art. no. 2048302.

[23] J. Zhang, C. Chen, Y. Xiang, W. Zhou, and A. V. Vasilakos, "An effective network traffic classification method with unknown flow detection," *IEEE Trans. Netw. Serv. Manage.*, vol. 10, no. 2, pp. 133–147, Jun. 2013.

[24] T. Groleat, M. Arzel, and S. Vaton, "Hardware acceleration of SVM-based traffic classification on FPGA," in *Proc. Intl. Wireless Commun. Mobile Comput. Conf.*, 2012, pp. 443–449.

[25] Y. R. Qu and V. K. Prasanna, "Enabling high throughput and virtualization for traffic classification on FPGA," in *Proc. IEEE 23rd Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, 2015, pp. 44–51.

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

[27] F. Gringoli, L. Nava, A. Este, and L. Salgarelli, "MTCLASS: Enabling statistical traffic classification of multi-gigabit aggregates on inexpensive hardware," in *Proc. 8th Int. Wireless Commun. Mobile Comput. Conf.*, 2012, pp. 450–455.

[28] D. Tong, L. Sun, K. Matam, and V. Prasanna, "High throughput and programmable online trafficclassifier on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 2013, pp. 255–264.

[29] J. R. Quinlan, *C4.5: Programs for Machine Learning*, S. L. Salzberg, Ed. San Mateo, CA, USA: Morgan Kaufmann Publishers, 1993.

[30] D. Angevine and N. Zincir-Heywood, "A preliminary investigation of skype traffic classification using a minimalist feature set," in *Proc. 3rd Int. Conf. Availability, Rel. Sec.*, Mar. 2008, pp. 1075–1079.

[31] N. W., S. Z., and G. A., "A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 5, pp. 5–16, 2006.

[32] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, 2009.

[33] "Tstat traces." [Online.] Available: Tstat Traces http://tstat.tlc.polito.it/traces.shtml

[34] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli, "Revealing skype traffic: When randomness plays with you," in *Proc. Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2007, pp. 37–48.

[35] J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms," in *Proc. SIGCOMM Workshop Mining Netw. Data*, 2006, pp. 281–286.

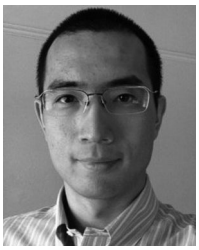
[36] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *Proc. IEEE Conf. Local Comput. Netw. 30th Anniversary*, Nov. 2005, pp. 250–257.

[37] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. 14th Int. Joint Conf. Artificial Intell.-Vol. 2*, 1995, pp. 1137–1143.

[38] Y.-S. Lim, H.-C. Kim, J. Jeong, C.-K. Kim, T. T. Kwon, and Y. Choi, "Internet traffic classification demystified: On the sources of the discriminative power," in *Proc. 6th Int. Conf.*, 2010, pp. 9:1–9:12.

[39] U. Fayyad and K. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *Proc. 13th Int. Joint Conf. Uncertainly Artificial Intell.*, 1993, pp. 1022–1029.

[40] Zynq-7000 All Programmable SoC, 2016. [Online.] Available: http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html



Da Tong received the BS degree in physics from Shanghai Jiao Tong University, in 2009 and the PhD degree in computer engineering from the University of Southern California, in 2016. His current research include online traffic classification for network routers and streaming algorithm and architecture for network anomaly detection. He is also interested in high performance data plane algorithms and architectures for Software Defined Networking (SDN) on various computing platforms such as multi/many core system, FPGA, and heterogeneous platforms. He is a member of the IEEE.

Yun Rock Qu received the BS degree in electrical engineering from Shanghai Jiao Tong University, in 2009, the MS degree in electrical engineering from the University of Southern California, in 2011, and the PhD degree in computer engineering from the University of Southern California, in 2015. His research focuses were multi/many-field packet classification, and online traffic classification. He is currently working in the Xilinx Labs at San Jose, California. His research interests include algorithm design, algorithm mapping onto custom hardware, high-performance and power-efficient architectures, and productivity language for heterogeneous system prototyping. He is a member of the IEEE and the ACM.

Viktor K. Prasanna received the BS degree in electronics engineering from Bangalore University, the MS degree from the School of Automation, Indian Institute of Science, and the PhD degree in computer science from Pennsylvania State University. He is Charles Lee Powell Chair in engineering in the Ming Hsieh Department of Electrical Engineering and professor of computer science with the University of Southern California. His research interests include high performance computing, parallel, and distributed systems, reconfigurable computing, and smart energy systems. He serves as the director of the Center for Energy Informatics at USC. He served as the editor-in-chief of the *IEEE Transactions on Computers* during 2003-06. Currently, he is the editor-in-chief of the *Journal of Parallel and Distributed Computing*. He was the founding chair of the IEEE Computer Society Technical Committee on Parallel Processing. He is the steering co- chair of the IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS) and is the steering chair of the IEEE/ACM International Conference on High Performance Computing (HiPC). He is a recipient of the 2009 Outstanding Engineering Alumnus Award from the Pennsylvania State University. He received the 2015 IEEE Computer Society W. Wallace McDowell award. He is a fellow of the IEEE, the ACM, and the American Association for Advancement of Science (AAAS).