High-Throughput Hash-based Online Traffic Classification Engines on FPGA*

Vaibhav R. Gandhi†
Dept. of Electrical Engineering,
Indian Institute of Technology, Gandhinagar
Gujarat, India 382424
vaibhavgandhi@iitgn.ac.in

Yun R. Qu, Viktor K. Prasanna
Ming Hsieh Dept. of Electrical Engineering,
University of Southern California
Los Angeles, CA 90089
yunqu@usc.edu, prasanna@usc.edu

Abstract—Traffic classification is used to perform important network management tasks such as flow prioritization and traffic shaping/pricing. Machine learning techniques such as the C4.5 algorithm can be used to perform traffic classification with very high levels of accuracy; however, realizing high-performance online traffic classification engine is still challenging. In this paper, we propose a high-throughput architecture for online traffic classification on FPGA. We convert the C4.5 decision-tree into multiple hash tables. We construct a pipelined architecture consisting of multiple processing elements; each hash table is searched in a processing element independently. The throughput is further increased by using multiple pipelines in parallel. To evaluate the performance of our architecture, we implement it on a state-of-the-art FPGA. Post-place-and-route results show that, for a typical 128-leaf decision-tree used for online traffic classification, our classification engine sustains a throughput of 1654 Million Classifications Per Second (MCPS). Our architecture sustains high throughput even if the number of leaves in the decision-tree is scaled up to 1K. Compared to existing online traffic classification engines on various platforms, we achieve at least 3.5 × speed-up with respect to throughput.

keywords - traffic classification, FPGA, hash tables

I. INTRODUCTION

Internet traffic classification is used to prioritize traffic flows (such as VoIP from video from email) and provide the appropriate Quality of Service (QoS) guarantees to them. Apart from optimizing current network operations, traffic classification is also important for planning improvements in future network architectures [1]. Today’s traffic classification engines need to be able to cope up with the burgeoning data rate of internet traffic and thus, there is a need for fast and accurate traffic classification in high-performance routers.

There are four categories into which the existing traffic classification approaches fall: (1) Port-number-based approaches classify traffic on the basis of the transport layer port numbers, a method that has become unreliable in the wake of dynamic assignment of port numbers by many applications today; (2) Deep Packet Inspection or DPI techniques compare the traffic payload against a predetermined set of signatures. These can by their very nature yield high efficiency [2], [3] but at the same time face privacy issues; (3) Heuristic-based techniques [4] classify traffic based on connection patterns, but suffer from low accuracy; (4) Machine Learning (ML)-based techniques use statistical properties of the traffic for classification and yield a high level of accuracy. The C4.5 algorithm [1], [5], [6] has been recognized as one of the most accurate ML techniques for traffic classification. However, realizing high-performance ML-based online traffic classification is still very challenging.

In this paper, we implement a high-throughput architecture for online traffic classification. The proposed architecture is based on a conversion from a C4.5 decision-tree to multiple hash tables. Specifically, our contributions include:

• We present an algorithm to convert a C4.5 decision-tree used in online traffic classification into a set of hash tables. The conversion technique can be extended for any generic decision-tree.
• We create modular Processing Elements (PEs) to perform search on these hash tables. We map these PEs onto a pipelined architecture to achieve high throughput while maintaining high resource efficiency.
• We achieve 1654 MCPS throughput for a typical decision-tree consisting of 128 leaves by implementing our architecture on a state-of-the-art FPGA. Our throughput is 3.5 × the throughput achieved by the state-of-the-art implementation of the C4.5 decision-trees on FPGA.

The rest of the paper is organized as follows: Section II covers the background work. In Section III, we introduce the algorithm for converting a decision-tree into hash tables. We present the architecture of the pipeline in Section IV. Section V evaluates the performance, and Section VI concludes this paper.

II. BACKGROUND AND RELATED WORK

A. C4.5 Algorithm

C4.5 algorithm, like all the ML-based traffic classification techniques, uses the statistical properties of traffic flows to classify internet traffic. Among a host of other ML algorithms like Naive Bayesian [7], K-means [8] and Support Vector Machine (SVM) [9]–[11], the C4.5 algorithm has demonstrated a high level of accuracy [6].

In [6], the effectiveness of multiple machine learning algorithms including SVM [9], [11], Naive Bayesian [7] and the
C4.5 algorithm [6] in classifying internet traffic are evaluated. The C4.5 algorithm achieves over 83% accuracy in identifying SSH traffic and over 97% accuracy in identifying Skype traffic, the highest among all five. In [1], nine flow level feature sets are evaluated with the conclusion that packet size, port number, and their related statistics achieve the highest accuracy. They also claim that C4.5 algorithm achieves the highest accuracy among the four algorithms they consider [6], [7], [9], [12].

B. Traffic Classification on FPGA

Accelerating online traffic classification using reconfigurable hardware (FPGA) has been the topic of many works. In [13], a high-performance FPGA-based traffic classification engine is presented that uses the C4.5 decision-tree. The performance of the classification engine is improved by efficiently storing it on hardware and by parallelizing memory accesses. Analytically, their architecture reduces the number of worst-case memory accesses.

In [14], FPGA and multi-core implementations of SVM-based online traffic classification are presented. It is shown that the hardware implementation conclusively outperforms the software implementation. [12] presents a multimedia traffic classification engine on FPGA that use the k-Nearest-Neighbour (k-NN) algorithm. Their architecture sustains a throughput of 250 MCPS. However, the classification accuracy relies on the amount of training data; a large amount of training data is required for high classification accuracy.

In [15], two architectures are presented optimized for high throughput and low hardware cost respectively. They directly map the C4.5 decision-trees onto FPGA and achieve a throughput of 520 MCPS. Each decision-tree level is mapped onto a pipeline stage. However, their design suffers from a longest wire length of $O(K)$, where $K$ is the number of nodes of a balanced decision-tree. Thus, for increasing tree size, the wire length increases; this affects negatively the clock-rate and the throughput. In this paper, we denote this implementation as the state-of-the-art implementation for decision-trees.

C. C4.5 Decision Tree

The C4.5 algorithm constructs a binary decision-tree with $N$ leaf nodes which can be described as follows: A field is a data set defined on a group of packets (e.g. destination port number, average packet size, etc.). The inputs to the C4.5 decision-tree are unique values in one or more fields after the discretization process [1]. Each node of the decision-tree represents a decision that has to be made depending on certain criterion in a particular field.

Figure 1 shows an example of a binary decision-tree that classifies traffic using two fields, the Source Port (SP) number and the Average Packet Size (APS). The decision-tree classifies traffic into four application classes - Skype, MSN, Thunder and HTTP. Accordingly, the leaf nodes represent one of these four classes. The non-leaf nodes represent a condition in one of the fields, in this case either SP or APS. The dashed-path shows the path to the leaf node “Thunder”; this leaf node can be reached whenever the input values of SP and APS satisfy $SP = 76$, $APS \neq 1000$ and $APS \neq 80$.

D. Problem Definition

We define the problem as: Given a binary decision-tree $T$ consisting of $N$ leaf nodes (indexed from left to right as $n = 0, 1, \ldots, N-1$ and $M$ fields (each with a width of $W_m$ bits, $m = 0, 1, \ldots, M-1$), design a high-throughput online traffic classification engine on FPGA.

We propose a novel architecture that converts C4.5 decision-tree used for traffic classification into a set of data structures (hash tables) that are suitable for mapping onto an FPGA. This conversion is discussed in the following section.

III. CONVERSION OF A DECISION-TREE TO HASH TABLES

Our approach to build the traffic classification engine entails the conversion of a decision-tree into a set of hash tables. These hash tables enable us to search each field individually and thus build modular processing elements that can be implemented in a pipeline yielding a high throughput.

The conversion of a decision-tree into hash tables outlined in Algorithm 1 can be understood in three phases:
1: finding paths from root node to all the leaf nodes;
2: finding the $N$-bit wide bit vectors corresponding to all the unique values associated with all the fields;
3: finding the hash functions and populating the hash tables with the bit vectors found in Phase 2.

A. Finding Root-to-leaf Paths

In a binary decision-tree, $T$, we can find $N$ paths starting from the root node to all the $N$ leaf nodes. With each node from the root node to the penultimate node there is associated a boolean condition. This condition is based on one of the known discrete values of one of the $M$ fields. Thus, with each node along such a path we “AND” a condition to the set of conditions seen by the path till that point. We call the set of all such conditions seen on the path from the root node to the $n$-th leaf node as the path-condition-set or $P_n$, where $n = 0, 1, \ldots, N-1$ is the index for all the leaf nodes from left
to be occupied then we pass the key through the second hash function. In either case, the hash value is obtained as the memory location to store the hash key.

2) If the second memory location is also occupied, we start over with a memory of depth $2 \times (U_m + 1)$. We repeat the above procedure till such time we are able to populate the hash table without any collision.

We illustrate the above procedure by constructing the hash table for the $SP$ field in our example tree from Figure 1. There are two unique values in the $APS$ field making $U_1 = 2$. We set out with memory of size $2 \times U_1 = 4$ and two hash functions as $key\%2$ and $key\%2 + 2$. After passing the unique value (hash key) 80 through the first hash function we get the index to be 0. So, the bit vector corresponding to $APS = 80$ is stored in memory at location 0. The next key 1000 is passed through the first hash function to get the index 0. But the memory

Algorithm 1 Conversion of Decision-Tree to Hash tables

**Input** A decision-tree $DT$ of $N$ leaves and $M$ fields.

**Output** $M$ hash tables for $M$ fields, each hash table consisting of multiple $N$-bit wide bit vectors.

1: On the $n$-th root-to-leaf path, $P_n$ denotes the path-condition-set, and $v_{n,i}$ denotes the $i$-th unique value
2: $F_m$ denotes the set of unique values in the $m$-th field
3: $BV_m$ denotes the bit vector in the $m$-th field
4: $U_m$ denotes the number of unique values in the $m$-th field
5: for $n = 0$ to $(N - 1)$ do
6: Find the path from the root node to the $n$-th leaf node
7: At each node:
8: if $v_{n,i} \notin F_m$ then
9: $F_m = F_m \cup v_{n,i}$
10: end if
11: $P_n = P_n \cap$ (condition at the node)
12: end for
13: for $m = 0$ to $(M - 1)$ do
14: for Each unique value $v_{n,i} \in F_m$ do
15: For each path $P_n$:
16: if $P_n$ has positive condition on $v_{n,i}$ or has negative conditions but not for $v_{n,i}$ then
17: Set the $n$-th bit of $BV_m = 1$
18: end if
19: end for
20: end for
21: for $m = 0$ to $(M - 1)$ do
22: Take hash table memory depth, $size = 2 \times U_m$
23: $index = v_{n,i} \% U_m$
24: if RAM[index] is not empty then
25: $index = index + U_m$
26: if RAM[index] is not empty then
27: $U_m = U_m + 1$
28: go to Step 22
29: end if
30: end if
31: end for
IV. ARCHITECTURE

We now map the hash tables discussed in Section III-C onto FPGA. To handle each field and its hash table, we present the modular Processing Element (PE) in Section IV-A. We show the overall architecture in Section IV-B.

A. Modular PE

In Figure 2, we also show the operations performed by a modular PE. The inputs to the modular PE are unique values in a field and the bit vector output from the previous PE. The PE applies the hash function on the unique values (hash keys) and finds the memory locations where the bit vectors are stored. The bit vector corresponding to a particular key is then read out from the memory; this bit vector is merged with the bit vector output from the previous PE by bitwise “AND” operation. The merged bit vector is passed onto the next PE.

Figure 3 shows the internal organization of a modular PE. In order to make better use of the memory resources, we use dual-port RAM. Thus, we have two keys and two bit vectors as inputs to the modular PE. Note:

• For a given key, an additional RAM named RAM_FLAG stores flags indicating whether to use the first or the second hash function.
• The contents in RAM_FLAG are compared with the keys. The comparison result directly drives the SELECT pin of the MUX.
• The function of the MUX is to select the index applicable for the given key out of the two available candidates as explained in Section III-C.
• The selected index is used to read the bit vector; this bit vector is then merged with the bit vector output by the previous PE.

B. Pipelined Architecture

The PE modules described in the foregoing section are mapped onto a pipeline as shown in Figure 4. The input to the pipeline is a set of unique values (input unique values), shown in the figure by Data_in. The input values in a particular field are passed onto the corresponding PE. The output of the pipeline is the leaf node matching the given set of input unique values. Several of these pipelines are set up in parallel to achieve a high throughput.

V. PERFORMANCE ANALYSIS

We conducted our experiments using Xilinx Vivado 2014.1 Design Suite, targeting the Xilinx Virtex-7 XC7VX980t FFG1930-2 FPGA. The device has 900 available bonded I/O pins and can be configured for large amount of distributed RAM (distRAM, up to 13 Mb) [16]. We constrain the clock rate to 200 MHz. We use the post-place-and-route results to report the maximum available clock rate and the resource consumption.

Our design is parameterized for the number of leaf nodes, the number of unique values in each field, and the number of pipelines. The 7 fields used to classify traffic as listed in Table I. They demonstrate high classification accuracy while
keeping hardware complexity reasonable [1], [5], [6]. We vary the number of fields between 6 and 9 to check the scalability of our design. The performance metrics for our analysis are:

- **Throughput**: The number of classifications performed by the engine per second measured in units of MCPS.
- **Resource consumption**: The total amount of hardware resources such as logic slices and I/O pins consumed on FPGA.

Note that we use dual-port memory which allows two concurrent classifications per pipeline and thus, a throughput that is twice the maximum clock rate with no increment in memory usage. Also, we use multiple pipelines in parallel. So, the actual throughput is twice the number of pipelines times the maximum achievable clock rate on FPGA.

We use distRAM instead of block RAM in our design. This localizes the memory accesses resulting in higher throughput.

### A. Throughput

We use our parameterized design to measure throughput against various decision-tree sizes. The number of leaf nodes are varied between 128 and 1024 so as to check the scalability of our design. The number of pipelines is varied between 1 and 3 as shown in in Figure 5. In Figure 5, we keep the number of fields fixed at 6. It is noted that the peak throughput for a decision-tree with 128 leaf nodes is 1654 MCPS. The throughput falls off by an average of 11.5% as the number of leaf nodes increases from 128 to 1024.

Figure 6 shows the variation in the throughput observed when the number of fields is varied between 6 and 9. Here, the size of the decision-tree is fixed to 128 leaf nodes and the number of parallel pipelines are 3. We observe that the throughput falls from 1654 MCPS to 1399 MCPS or by 15.4% as the number of fields increases from 6 to 9. Our design is thus scalable with respect to both the size of the decision-tree and the number of fields used for classification.

### B. Resource Consumption

Logic slices and I/O are precious FPGA resources and it is crucial that their utilization be carefully studied. The amount of these resources used in one pipeline dictates the maximum number of pipelines that can be created and thus the maximum throughput that can be achieved on FPGA. The resource usage also affects power consumption of the classification engine. Thus, it is important that this usage be parsimonious.

Table II shows the number of logic slices and I/O pins used by the designs of various parameters. The design uses distRAM only leaving the block RAM to be unused. Note that logic slices are relatively plentiful whereas the real paucity is in the number of I/O pins. For each pipelines greater than 25% of bonded I/O pins are utilized, limiting the number of pipelines to 3. This can however be remedied by providing all the input unique values serially.

<table>
<thead>
<tr>
<th>No. of pipelines</th>
<th>N</th>
<th>Logic slices</th>
<th>I/O (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>393</td>
<td>26.88</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>417</td>
<td>26.88</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>448</td>
<td>26.88</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>501</td>
<td>26.88</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>780</td>
<td>51.77</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>837</td>
<td>51.77</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>879</td>
<td>51.77</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>1002</td>
<td>51.77</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>1191</td>
<td>76.66</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>1255</td>
<td>76.66</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>1350</td>
<td>76.66</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>1507</td>
<td>76.66</td>
</tr>
</tbody>
</table>

### C. Comparison with Existing Classifiers

Comparing our classification engine with the state-of-the-art implementation [15], we achieve $3.5 \times$ improvement with respect to throughput. Figure 7 shows the comparison results with respect to various numbers of decision-tree leaf nodes. Note their implementation was based on Virtex-6 devices, while their throughput (in Gbps) was measured for a group of five consecutive 40-byte packets. The following observations can be made:

- Our hash-based technique consumes significantly less logic resources compared to [15]. For 1024 leaf nodes and...
TABLE III: Comparison with existing approaches

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Algorithm</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jiang et al. [12]</td>
<td>k-NN</td>
<td>Post-place-and-routed on Virtex-5 FPGA</td>
</tr>
<tr>
<td>Groleat et al. [14]</td>
<td>SVM</td>
<td>Synthesized on Virtex-5 FPGA</td>
</tr>
<tr>
<td>Tong et al. [15]</td>
<td>C4.5</td>
<td>Post-place-and-routed on Virtex-6 FPGA</td>
</tr>
<tr>
<td>Our work</td>
<td>C4.5</td>
<td>Post-place-and-routed on Virtex-7 FPGA</td>
</tr>
</tbody>
</table>

For small decision-trees, the throughput for our single pipeline case is only marginally higher than that of the architecture presented in [15]. However, as the number of decision-tree leaf nodes increases, the throughput of our design decreases much slower. Also, high resource efficiency means that the potential number of concurrent pipelines can be high in our architecture.

The foregoing observations suggest that our implementation is superior in terms of scalability than the design presented in [15]. We show in Figure 8 the throughput comparison of our design with existing traffic classifiers. Table III summarizes platforms and algorithms used by these approaches. As can be seen, our design achieves consistently better throughput.

VI. CONCLUSION

In this paper, we presented an algorithm to convert a C4.5 decision-tree into a set of hash tables avoiding any collisions. We created modular processing elements to search each field using hashing. The design was mapped onto a state-of-the-art FPGA. We evaluated the performance of the architecture and demonstrated three key attributes: modularity, high throughput and high resource efficiency. The high throughput of our design essentially stems from the hash-based search performed in our architecture.

In the future, we plan to apply the hashing modules developed to other problems such as packet classification. The conversion technique presented in this paper can also be generalized for other generic classification problems. For our design, a thorough comparison among various platforms is also of great interest.

REFERENCES