High Throughput and Programmable Online Traffic Classifier on FPGA*

Da Tong, Lu Sun, Kiran Kumar Matam, Viktor Prasanna
Ming Hsieh Department of Electrical Engineering
University of Southern California
{datong, lusun, kmatam, prasanna}@usc.edu

ABSTRACT

Machine learning (ML) algorithms have been shown to be effective in classifying the dynamic internet traffic today. Using additional features and sophisticated ML techniques can improve accuracy and can classify a broad range of application classes. Realizing such classifiers to meet high data rates is challenging. In this paper, we propose two architectures to realize complete online traffic classifier using flow-level features. First, we develop a traffic classifier based on C4.5 decision tree algorithm and Entropy-MDL discretization algorithm. It achieves an accuracy of 97.92% when classifying a traffic trace consisting of eight application classes. Next, we accelerate our classifier using two architectures on FPGA. One architecture stores the classifier in on-chip distributed RAM. It is designed to sustain a high throughput. The other architecture stores the classifier in block RAM. It is designed to operate with small hardware footprint and thus built at low hardware cost. Experimental results show that our high throughput architecture can sustain a throughput of 550 Gbps assuming 40 Byte packet size. Our low cost architecture demonstrates a 22% better resource efficiency than the high throughput design. It can be easily replicated to achieve 449 Gbps while supporting 160 input traffic streams concurrently. Both architectures are parameterizable and programmable to support any binary-tree-based traffic classifier. We develop a tool which allows users to easily map a binary-tree-based classifier to hardware. The tool takes a classifier as input and automatically generates the Verilog code for the corresponding hardware architecture.

Categories and Subject Descriptors

C.1.3 [**Computer Systems Organization**]: PROCESSOR ARCHITECTURES—*Adaptable architectures*

*This work was partially supported by the US NSF under grant CCF-1116781.

FPGA'13, February 11–13, 2013, Monterey, California, USA.

General Terms

Algorithms, Design, Performance

Keywords

Traffic classification, machine learning, C4.5 decision tree, discretization, FPGA acceleration, high throughput, programmable, low cost

1. INTRODUCTION

Traffic classification forms basis for many important network management tasks such as flow prioritization, traffic shaping/policing, and diagnostic policing. Accurate traffic classification also benefits network security in dynamic access control and intrusion detection. In addition to being an essential part in network operations, traffic classification also contributes to the planning of future network architecture [13, 5, 21].

Existing traffic classifiers fall into four categories: (1) *port number* based schemes classify traffic based on well-known transport layer port numbers; (2) *deep packet inspection* (DPI) examines the traffic payload against a set of known signatures; (3) *heuristic* based techniques [10, 11] classify traffic based on connection patterns; and (4) *machine learning* based techniques exploit statistical properties of traffic. However, port number based approaches are no longer reliable because applications today tend to use dynamic port numbers for connections. DPI techniques [18, 16] can reach extremely high accuracy; but they cannot cope with encrypted traffic. Heuristic based approaches suffer from low accuracy and large memory requirement to store the connection patterns. On the other hand, machine learning based traffic classification has drawn a lot of attention in the research community recently [3, 1, 13]. It demonstrates not only high accuracy but also robustness to today's dynamic internet traffic. However, realizing such classifiers to meet high data rates is challenging. Hardware acceleration for these classifiers is required to achieve online classification.

In this paper, we propose two architectures to realize complete online traffic classifier using *flow-level* features. These two architectures use distributed RAM and block RAM respectively. We conduct detailed experiments to evaluate the performance of these architectures in terms of throughput, resource efficiency and scalability on a state-of-art FPGA device.

Our contributions in this work are:

- We conduct extensive experiments to identify a feature set that achieves high accuracy. Our classifier

built upon this flow-level feature set can achieve an overall accuracy of 97.92% in classifying a traffic trace consisting of eight application classes.

- We propose complete hardware solution to machine learning based traffic classifier. Our architectures not only realize online traffic classification, but also perform online discretization which is an essential step in any machine learning based technique.

- To the best of our knowledge, our high throughput architecture is the first FPGA design for 400+ Gbps online traffic classification using *flow-level* features. Implementation results on a state-of-the-art FPGA show that our high throughput design can achieve a throughput of 550 Gbps. On the other hand, our low cost design demonstrates a 22% better resource efficiency than the high throughput design. It can be easily replicated to achieve 449 Gbps while supporting 160 input traffic streams concurrently.

- Both architectures are parameterizable and programmable to support any binary-tree-based traffic classifier.

The rest of the paper is organized as following. Section 2 gives the background and related work in traffic classification. Section 3 discusses the methodology to select a high accuracy feature set. Section 4 describes the C4.5 decision tree based classifier. Section 5 presents the two hardware architectures to accelerate the classifier. Section 6 evaluates our prototype implementation on FPGA. Section 7 concludes the paper.

2. RELATED WORK

2.1 C4.5 based Traffic Classification

Machine learning based techniques classify internet traffic based on statistical properties of traffic flows [3, 1, 13]. Among existing machine learning approaches, C4.5 algorithm has been recognized as one of the most accurate algorithms for traffic classification.

[1] evaluates the effectiveness of machine learning algorithms in classifying encrypted traffic. They study five Algorithms AdaBoost, Support Vector Machine(SVM), Naive Bayesian, RIPPER, and C4.5 Algorithm. They use a set of 22 features, including packet-level features (e.g. packet sizes) and flow-level features (e.g. packet inter-arrival time), to build the machine learning classifiers. They show that C4.5 decision tree Algorithm gives the best accuracy among all the considered techniques. They achieve over 83% accuracy in identifying SSH traffic and over 97.8% accuracy in identifying Skype traffic.

[13] investigates the discriminative power of nine flow-level feature sets. They show that features such as packet size, port number and their related statistics always give the highest accuracy. This observation holds for all the four machine learning Algorithms they evaluate, i.e., Naive Bayes, SVM, k-Nearest-Neighbour(kNN), and C4.5 decision tree. They claim that C4.5 Algorithm gives the highest accuracy using any feature set. They also show that discretization is a necessary preprocessing step in traffic classification. The accuracy of machine learning Algorithms can be greatly improved by using discretized training dataset.

[2] shows that Skype traffic can be accurately classified without investigating packet payload. They evaluate two classifiers built from AdaBoost and C4.5 using ten flow-level features. AdaBoost classifier achieves an accuracy of 97% when classifying UDP Skype traffic and 94% accuracy when classifying TCP Skype traffic. The C4.5 classifier can achieve an accuracy of 98% and 94% respectively.

2.2 FPGA based Traffic Classifier

In order to accelerate the machine learning based traffic classifier, several works have been proposed to implement the classifiers in hardware.

In [14], an FPGA based architecture is presented to improve the classification performance of C4.5 decision tree classifier. They reduce the number of memory accesses during the search by efficiently storing the classifier on hardware. Memory accesses are parallelized to further improve the performance. They analytically show that their architecture reduces the number of memory accesses in the worst case from 26 to 5. However, no FPGA implementation results of the architecture are presented in their paper.

[9] proposes an FPGA based architecture to realize an accurate and high throughput classification of multimedia traffic using simple *packet-level* features. Their classifier is built upon k-Nearest-Neighbor Algorithm and trained using three packet-level features (port number, packet size, and protocol). Their classifier achieves high accuracy when the number of training data exceeds $10K$. The place and route result of their hardware implementation shows that they achieve a throughput of 40 Gbps. As they consider only three simple packet-level features, their architecture is restricted to classifiers with small number of application classes. It cannot handle general classifier using flow-level features.

[15] proposes a hardware architecture on NetFPGA for decision tree based classifier. Their architecture is fully parameterizable in terms of throughput, number of features, tree depth and maximum number of nodes with in a tree level. So their architecture can support any general decision tree based classifier. Several optimizations on storage of the tree structure are proposed to improve latency and lower hardware cost. Although their paper presents a programmable implementation of a decision tree based classifier, their experimental results show that the clock rate they achieve is around 67 MHz only. This clock rate is not sufficient to perform online traffic classification.

We are not aware of any prior work that realizes a complete flow-level online traffic classifier. None of the previous works propose a hardware solution for discretizaion, which is an essential step in traffic classification. In this paper, we first develop a highly accurate classifier using flow-level features. We then propose two architectures to enable online discretization and classification. Both architectures are programmable with respect to the structure of the classifier, i.e., number of features, number of nodes in each level, number of application classes, and so on. They are flexible to support any binary-tree-based classifier.

3. CONSTRUCTION OF FEATURE SET

One of the challenge in building a machine learning based traffic classifier is to choose the appropriate set of features. Appropriate feature set is the set of features which are effective in classifying the targeted applications from each other.

Table 1: Application Protocols in Our Traffic Traces

Application Protocol	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

In this section we discuss the methodology for identifying the feature set.

3.1 Dataset

Our dataset is built from two sources: a general traffic trace provided by a major network vendor and Tstat [19], a publicly available labeled traffic trace. Unlike many previous works targeting at only one or two application classes, our dataset consists of eight application classes as listed in Table 1. These classes 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 class in the dataset consists of 700 traffic flows. This dataset is used as both training set and testing set in our experiments.

3.2 Feature Selection

3.2.1 Candidate Features

Unlike many previous works which focus only on accuracy, the features in our work must be feasible in an online traffic classifier. Therefore 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 application classes. *true positive rate* is the fraction of traffic flows correctly classified for an application class.
- Early Classification: It is important to decide the class 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: it requires high computation power to calculate statistical features from traffic flows. This limits the usage of these features in online traffic classifiers where features need to be computed very fast. So we constrain our selections to a set of low cost features. They are avg/min/max/variance of features.

In order to achieve high accuracy in classifying a traffic trace consisting of broad application classes, our work uses flow-level features as opposed to packet-level features used in many other works. A flow is defined as a series of packets that share the same five tuple information: protocol, source port number, destination port number, source IP address and destination IP address. 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 [13, 2, 21, 1, 4, 3, 5]. They are

- *Protocol*: the protocol associated with the flow.
- *Src. Port Number*: the source port number associated with the flow.

Table 2: Candidate Feature Sets

Feature Set	Classic Features			Sizes of First N Packets	Packets size statistics			
	Protocol	Src. Port No.	Dst. Port No.		Avg. Packet Size (byte)	Max. Packet Size (byte)	Min. Packet Size (byte)	Var. of Packet Size
Set A	×	×	×					
Set B	×	×	×	×				
Set C	×	×	×		×	×	×	
Set D	×	×	×		×	×	×	×
Set E	×	×	×	×	×	×	×	
Set F	×	×	×	×	×	×	×	×

- *Dest. Port Number*: the destination port number associated with the flow.
- *Sizes of First N Packets*: sizes of the first N packets in the flow.
- *Avg. Packet Size*: average packet size of the first N packets in the flow.
- *Max. Packet Size*: maximum packet size among the first N packets in the flow.
- *Min. Packet Size*: minimum packet size among the first N packets in the flow.
- *Var. of Packet Size*: variance of sizes of the first N packets in the flow.

We group the above features into six sets as shown in Table 2. 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; *Avg/Max/Min/Var Packet Size* as statistical features. The statistical features listed above comes from first N packets in the flow. Next, we will identify the best value of N and the appropriate set of features for our classifier.

3.2.2 Methodology

We conduct extensive experiments to identify the value of N and the appropriate feature set for our classifier. Each feature set is evaluated using the same machine learning Algorithm (i.e., C4.5 Algorithm) and dataset. We apply the commonly used 10-fold cross validation technique [12] which breaks the dataset into 10 equally sized sets. In this technique, we perform 10 iterations; during each iteration, 9 sets are used as training sets and 1 set is used as 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.

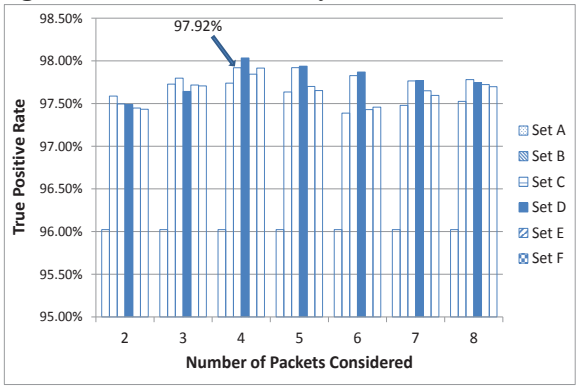
All training and testing data are discretized in the experiments. Discretization is the process of converting continuous feature values into discrete feature values. It is known that discretization can help to create more accurate machine learning classifiers. Previous literature [13] has also shown that this fact holds true when applying machine learning Algorithms to traffic classification. We adopt Entropy-MDL [6], the most commonly used discretization Algorithm, to convert feature values into discrete values.

3.2.3 Emperically Optimized Feature Set

In this section we use the methodology discussed in Section 3.2.2 to identify a set of features that achieves the highest accuracy. We denote this emperically optimized feature set (EOFS). Note that, our methodology explores a limited design space using the available training data.

Figure 1 shows the overall accuracy achieved by each feature set. It can be observed that Feature Set C and Feature Set D achieve the highest accuracy among all feature sets.

Figure 1: Overall Accuracy of Each Feature Set



Feature Set C and Feature Set D differs only by one feature, i.e. variance of packet size. Computing variance requires more hardware resources; therefore, Feature Set C is favored over Feature Set D.

From Figure 1 it can also be observed that maximum overall accuracy is reached when using features from the first 4 packets in a flow. Having fewer than 4 or more packets in the feature set decreases the overall accuracy. This is because our traffic trace includes P2P and IM flows whose first 3-4 packets in a flow are used for initial connection establishment. Thus the values and statistical properties of the first few packets are essential in classifying these traffic flows.

To conclude, Feature Set C is our EOFS and it is computed from first 4 packets in a traffic flow. It achieves an overall accuracy of 97.92%.

3.2.4 *Pitfalls of Classic Feature Set*

Our evaluation also demonstrates that the classic feature set (i.e., $Protocol$, $Src.PortNumber$, and $Dest.PortNumber$) used in traditional machine learning based traffic classification scheme is not sufficient in classifying a general traffic trace where P2P traffic exists. Although the overall accuracy achieved by the classic feature set reaches 96.0%, it cannot distinguish different classes of P2P traffic from each other.

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

It can be observed from Table 3 that the major pitfall of classic feature set is that it cannot distinguish P2P traffic from each other. For example, 8% of the Thunder traffic has been falsely classified as Skype; 4.25% and 1.15% of the Skype traffic has been falsely classified as Thunder and P2PTV respectively. This directly degrades the overall accuracy of classic feature set shown in Figure 1.

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% of Skype traffic is falsely classified as Thunder traffic; only 1.60% of Thunder traffic is falsely classified as Skype traffic. We can expect that our EOFS outperforms the traditional classic feature set in a real world setting where more number of classes from P2P exist.

Table 3: Confusion Matrix for Classic Feature Set

	HTTP	MSN	P2PTV	QQ _IM	Skype	Skype _IM	Thun.	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	Thun.	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

4. C4.5 CLASSIFIER

As mentioned in Section 3, our traffic classifier is built upon C4.5 decision tree Algorithm. C4.5 Algorithm is a well-known machine learning algorithm proposed by Quinlan [17]. C4.5 based classifiers have demonstrated high true positive rate with different target applications, test traces, and experiment setups [1] [2] [20]. This section presents in detail how we apply C4.5 Algorithm to build our classifier. It consists of two steps: input discretization and classifier construction. Note that the constructed discretizer and the C4.5 classifier are implemented in hardware.

4.1 Discretization

In Section 3, we identify Feature Set C, i.e., {source port number, destination port number, average packet size, max packet size, and min packet size}, as the feature set for our classifier. We first discretize all numerical values of these features in the sample traffic flows before using them as training data to build the classifier. We adopted Entropy-MDL [6], the most commonly used discretization Algorithm, to convert these feature values into discrete values.

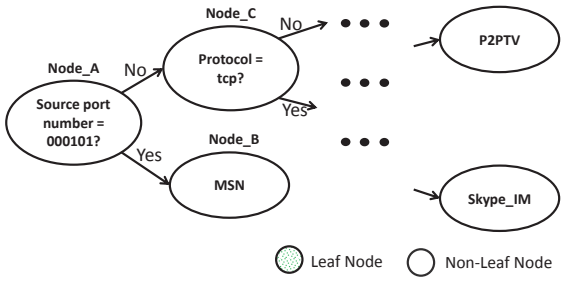
4.2 Building the C4.5 Classifier

The training data used in our work are the feature vectors extracted from labeled traffic flows. A feature vector consists of the discretized values of the six features in our feature set and a label indicating the application class associated with the flow.

We extracted in total 5600 training samples from our dataset into WEKA [7], an existing machine learning software. The 5600 training samples consist of 700 samples from each of the application classes in Table 1.

The classifier generated by the C4.5 Algorithm is essentially a binary decision tree. It can be characterized by leaf nodes and non-leaf nodes. Each leaf node in the decision tree is an application class. When an input feature vector arrives at a leaf node, it is classified as the application class associated with the leaf node. Each non-leaf node is a decision node which has a comparison key, a left child node and a right child node. The comparison key has a value and a feature type. When an input feature vector arrives at a non-leaf node, it is compared against the key at the node. If the feature value equals the key, the input feature vector

Figure 2: An Example of C4.5 Decision Tree Based Classifier



will be passed to the left child node; otherwise, it will be passed to the right child node.

Figure 2 shows a simple example of a C4.5 decision tree classifier. In this example, $Node\_A$ is a non-leaf node, its key has a value of 000101 for feature type 'source port number'. $Node\_B$ is its left child node and $Node\_C$ is its right child node. $Node\_B$ is a leaf node; any input feature vector that arrives at $Node\_B$ will be classified as application class of MSN.

4.3 Classification Algorithm

Once the C4.5 classifier is built, the classification of any test feature vector from a traffic flow consists of two phases: discretization and classification. The feature vectors in the test traffic flows are converted to discrete vectors during the discretization phase. During the classification phase, the discretized feature vectors traverse the C4.5 classifier until they reach a leaf node.

Let q denote an instance of test traffic flow. Our goal is to classify q into an application class. Let $F[i]$ denote the raw feature value of feature type i. Then, q has a raw feature vector consisting of six feature values {F[0], F[1], F[2], F[3], F[4], F[5]}, i.e., {protocol, source port, destination port, avg. packet size, max. packet size, and min. packet size}. Let {D[0], D[1], D[2], D[3], D[4], D[5]} denote the output discrete values from the discretization phase. These values are passed as an input feature vector into the classification phase. The detailed Algorithms involved in the discretization phase and the classification phase are shown in Algorithm 1 and Algorithm 2.

5. ARCHITECTURE

The proposed classification Algorithm in the previous section is highly desirable for hardware implementation. In this section, we propose a high throughput architecture and a low cost architecture to realize online classification of traffic flows in hardware. The high throughput architecture adopts a pipelined design and takes advantage of the on-chip distributed RAM to achieve high performance. The low cost design minimizes the usage of hardware resources by reusing the hardware components.

As stated in Section 4, an input traffic flow is represented by a feature vector consisting of raw values of its features, i.e., protocol, source port number, destination port number, avg. packet size, max. packet size, and min. packet size. First, it is passed through the discretizer which outputs a discrete feature vector. This discrete feature vector will then traverse the C4.5 tree classifier until it reaches a leaf node.

Algorithm 1 Phase 1: Discretization for each feature

Let $d_1, d_2, ..., d_{n-1}$ divide $[0, \infty]$ continuous space in to n intervals. Let i_1 denote $[0, d_1)$,i_2 denote $[d_1, d_2)$, i_3 denote $[d_2, d_3)$,...,i_{n-1} denote $[d_{n-2}, d_{n-1})$, and i_n denote $[d_{n-1}, \infty]$. Let $c_1, c_2, ...c_n$ denote the corresponding discretized values for $i_1, i_2, ..., i_n$.
F is the given input feature value for which we need to identify the discretized value.
D is the output discretized value for F.
Let $High = n, Low = 1$
while true **do**
  $Mid = (High + Low)/2$
  if $High$=Low **then**
    D=c_{High}, break
  end if
  if $F < d_{Mid}$ **then**
    $High$=Mid
  else
    Low=$Mid + 1$
  end if
end while

Algorithm 2 Phase 2: Classification

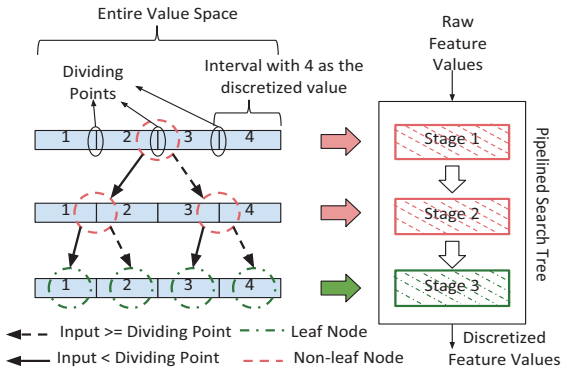
Let *current_node* be the node which is currently accessed.
Let *value* denote the value of the key stored at *current_node*. *value* is an application class in leaf nodes.
For non-leaf nodes, let $feature$ denote the type of the key stored in the node. Since there are six features involved in the classifier, $feature$ is an integer between 0 and 5.
Initialize *current_node* = *root node*.
while *current_node* $\neq$ *leaf node* **do**
  if $D[feature] = value$ **then**
    current_node = *left child node*.
  else
    current_node = *right child node*
  end if
end while
q = *value at current_node*.

5.1 Performance Metrics

This paper considers the following factors when constructing the classifier on hardware:

- *Throughput* is measured as the number of bits classified per second. The minimum raw *throughput* can be computed as the product of number of flows classified per second, minimum packet size (i.e., 40 Bytes), and number of packets needed in a traffic flow to extract features (i.e., 4 packets).
- *Resource Efficiency* is measured as amount of hardware resources per unit of throughput. On a FPGA device, the amount of hardware resources is represented by number of occupied slices. For a block RAM, we assume the hardware resources it consumes equals the number of slices of the same size. On a Virtext-6 FPGA, each slice contains four 6-input Look-Up Tables and each block RAM provides 18 Kb of memory. Therefore, one block RAM is equated to 72 slices. Since block RAM is highly optimized in terms of area, we multiply the number of slices it consumes with a

Figure 4: Mapping from Discretizer to Hardware



factor of 1/35. This is because it is observed that FPGA on average consumes 35 times more area than an ASIC platform which is highly optimized for area [8]. Therefore, the hardware resources consumed by one block RAM equals (72 slices) $\times$ *percentage of block RAM occupied* $\times$ 1/35.

5.2 High Throughput Design

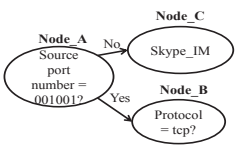
The mapping of the C4.5 classifier to hardware is shown in Figure 3. The classifier consists of a discretizer and a C4.5 tree classifier. Both components can be mapped to hardware as a binary tree. Each level of the binary tree corresponds to a stage in the hardware. All stages are pipelined to achieve high throughput. Each stage consists of a processing element and a distributed RAM. The distributed RAM stores all the nodes of the corresponding level in the tree. When a feature vector arrives at a node in the stage, the node is retrieved from the corresponding distributed RAM. Based on the content in the retrieved node, the processing element will decide the address of the node to be accessed by the feature vector in the next stage.

5.2.1 Discretizer

The discretization phase described in Section 4.3 can be realized by a binary search tree in hardware. Let $n-1$ dividing points divide the continuous space of the input values in to n intervals. The midpoint among the dividing points is stored at the root node of the binary search tree. All the intervals less than this dividing point are stored in the left subtree of the root node and remaining intervals in the right subtree of the root node. The left subtree and right subtree are recursively defined in the same way. Each leaf node stores the end points of an interval. Each level in the binary search tree is mapped to a pipeline stage in the hardware architecture. So the input value of a feature is discretized by traversing the binary search tree in a pipeline manner. An example on mapping of the discretizer to hardware architecture is illustrated in Figure 4.

In our architecture, all the tree nodes are stored in distributed RAM. Each stage of the pipeline consists of a distributed RAM and a processing element (PE). In the distributed RAM, the nodes of the corresponding level in the tree are stored. The PE decides whether the feature vector traverses to the left or right child node.

Figure 5: Example of Memory Structure



Node_A:

IsLeaf	feature_type	feature_value	address of left child node	address of right child node
0	001	001001	address of Node_B	address of Node_C

Node_C:

IsLeaf	class_ID
1	0110

5.2.2 Programmable Memory Organization

Nodes of a stage are stored in the distributed RAM attached to it. One memory word stores one node. Therefore, the number of entries in each distributed RAM equals the number of nodes in that level of the tree. In the discretizer, the memory word storing a node consists of the value of the dividing point and the addresses of its child nodes. In the C4.5 tree classifier, the memory word storing a leaf node consists of $classs\_ID$ associated with the node and a 1-bit mask indicating whether it is a leaf node. The memory word storing a non-leaf node consists of the same 1-bit mask, the type of feature associated with the key in the node, the value of the key, and the addresses of its child nodes.

Figure 5 gives an example of memory structure for a leaf node and a non-leaf node in a C4.5 tree classifier. $Node\_A$ is a non-leaf node which has a feature value of 001001. Its key type is source port number. In our classifier, this feature type is identified as 001. $Node\_C$ is a leaf node which is associated with the Skype_IM application class. In our classifier, this application class is identified by $class\_ID$ of 0110.

The structure of the memory storing the C4.5 tree classifier is fully parameterized. Number of memory words in the memory attached to a stage is parameterized with number of nodes in the corresponding tree level. Width of a memory word (i.e., number of bits representing a memory word) is the sum of widths of $feature\_type$, $feature\_value$, and addresses of child nodes. These widths are parameterized and can be determined from the structure of the C4.5 tree classifier. Number of feature types in the classifier determines the width of *feature_type*; range of discretized feature values determines the width of *feature_value*; number of child nodes in the next tree level determines the address width of child nodes. For a memory word storing a leaf node, number of applications classes in the classifier determines the width of *class_ID*.

These parameterizations enable the programmability of our architecture. Any binary-tree-based classifier can be mapped to our architecture by specifying the parameters mentioned above. Users can generate the memory structure suitable for their classifier by programming the parameters. Once the memory structure is generated, the classifier can be loaded into memory accordingly.

5.2.3 Processing Element in C4.5 Classifier

Figure 6 shows the high level architecture of processing element in each stage. The processing element takes as input a feature vector, a $class\_ID$, and the address of the node to be accessed. A node is first retrieved from the distributed RAM attached to the stage based on the input address. If a leaf node is retrieved, feature vector will be classified by assigning the value in leaf node to $class\_ID$. If a non-leaf node is retrieved, the input feature vector will be compared against the key in the node. The address of the node to

Figure 3: Mapping from Decision Tree to Hardware

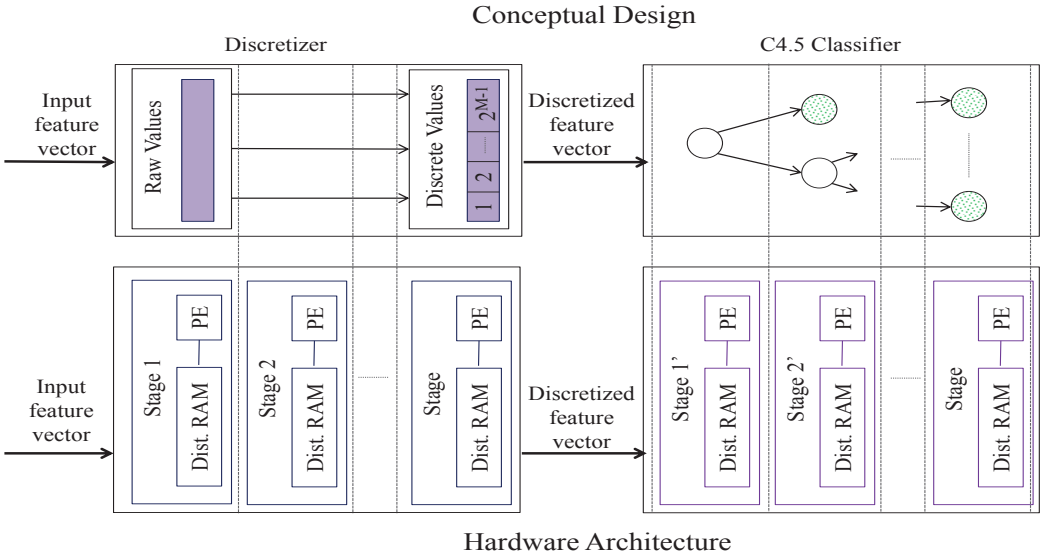
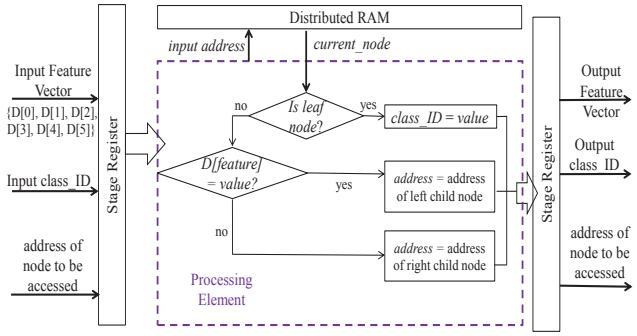


Figure 6: Architecture of Processing Element



be accessed in the next stage will be decided based on the comparison result.

5.3 Low Cost Design

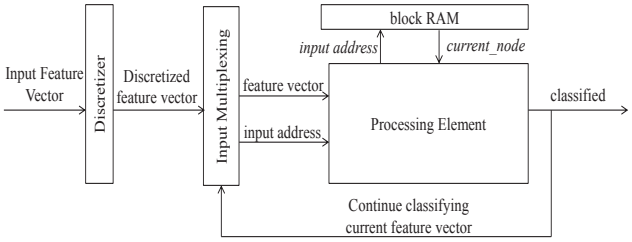
In addition to the high throughput architecture, we also propose an architecture which can be built at low hardware cost. This architecture is designed to operate with small hardware footprint. It stores the classifier in a block RAM. In contrast to the high throughput design where each level of the decision tree and discretizer is mapped to a separate Processing Element attached with a distributed RAM, the low cost design allows all levels of C4.5 tree classifier to share the same Processing Element and memory.

Figure 7 shows the overall architecture of the low cost design. The architecture consists of four components: Discretizer, Input Multiplexer, block RAM, and Processing Element. All tree nodes in the classifier are stored in block RAM. Since a block RAM is dual ported, our architecture can support two input traffic streams concurrently.

When an input feature vector comes in, it first goes through Discretizer. Then, it traverses the decision tree by iteratively going through Input Multiplexing, block RAM and Processing Element until it is classified.

The architecture is at low hardware cost in the sense that the logic it consumes does not grow with the size of the C4.5

Figure 7: Overall Architecture of Low Cost Design



tree classifier. In contrast to the high throughput architecture where each level of the classifier consumes one Processing Element, this architecture uses only one Processing Element and Input Multiplexer regardless of the size and number of levels of the classifier. Due to the low cost nature of this architecture, it can be duplicated to support multiple input traffic streams. Impact of number of input traffic streams on the throughput and resource usage of this architecture will be evaluated later in Section 6.3. The components in this architecture are described below.

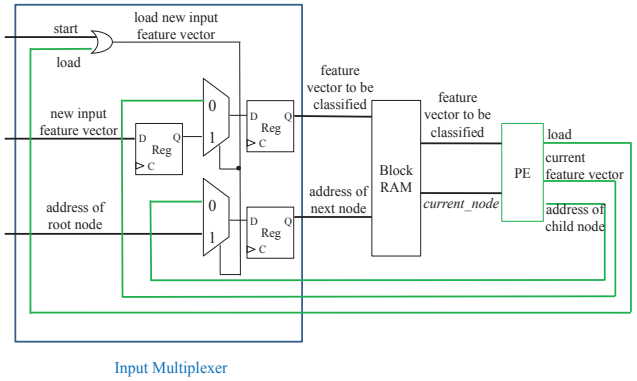
5.3.1 *Input Multiplexer*

The component decides whether a feature vector should continue traversing the C4.5 tree classifier or a new input feature vector should be loaded. Figure 8 shows the detailed architecture of the input multiplexing component. After a new input feature vector is loaded, it will pass the new vector to the next stage along with the address of the root node of the tree. When continuing on classifying the current feature vector, it will pass this current vector to the next stage along with the address of the child node to be accessed in the next stage. The current feature vector and the address of the child node are provided by the feature comparison unit.

5.3.2 *Block RAM and Processing Element*

This component retrieves the child node to be accessed from block RAM. The memory organization is defined in

Figure 8: Architecture of Input Multiplexer



the same way as in our high throughput design. The dual port block RAM can support two accesses concurrently.

The Processing Element logic is similar to that in our high throughput design with slight modification to fit into the low cost architecture. In addition to performing the logic mentioned in our previous design, the Processing Element in this low cost design also outputs a control signal. This control signal is passed to Input Multiplexer and is enabled when a feature vector reaches a leaf node. The Input Multiplexer will load a new input feature vector when this signal is enabled.

5.3.3 Low Cost Discretizer

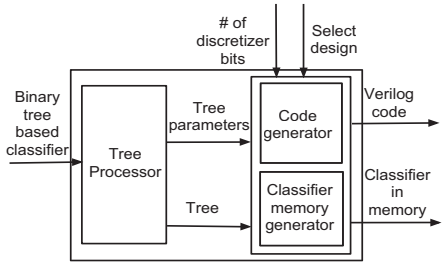
The low cost discretizer stores all tree nodes in one block RAM and makes usage of one processing element only. The input raw feature vector iteratively goes through the processing element until it is eventually discretized. Its architecture looks very similar to the low cost architecture of the C4.5 tree classifier described above.

5.4 Architecture Generation Tool

In our design, the operation of the PE does not depend on the classifier structure. However as explained in Section 5.2.2, width of memory words, number of memory words, and number of the levels of pipeline stages depend on the classifier. We parameterize these variables in our memory design. So our designs are programmable to support different binary tree based classifiers.

Our architecture generation tool facilitates users in generating the Verilog code. The tool accepts binary tree based classifier as input to our tool and generates Verilog source code. The tool consists of three modules: `Tree processor`, `Code generator`, and `Classifier memory generator`. The binary tree based classifier which is to mapped on to the hardware is given as input to the `Tree processor` module. Based on the binary tree based classifier, `Tree processor` module computes the tree parameters and passes them to the `Code generator` module. The parameters include 1) Number of feature types, 2) Number of levels in the tree, 3) Number of nodes in each level, and 4) Number of application types. Based on these tree parameters the `Code generator` module computes the design parameters and generates the verilog source code. The design parameters are 1) number of bits required by the fields in the memory word (feature_type, feature_value, class_ID, and the left and right pointers in the node), the number of levels of pipeline for the high throughput design.

Figure 9: Architecture generation tool



The binary tree based classifier is passed from the `Tree processor` module to the `Classifier memory generator` module which generates the tree in to memory. The generated tree is stored level by level. In each level the number of nodes in the level and the corresponding nodes are stored. The nodes are stored as explained in Section 5.2.2. Users can also select the number of output discretizer bits and the design to use among our two designs. The architecture of our tool is shown in Figure 9.

6. EXPERIMENTAL RESULTS

We implement both high throughput architecture and low cost architecture on FPGA. Our target device is Virtex-6 XC6LX670 with -2 speed grade. All results are post place and route results using Xilinx ISE 14.2.

Our architecture consists of a discretizer and a C4.5 classifier. Both the components are parameterized. The parameters for the discretizer are

- *width of raw feature value*: this is the number of bits representing the raw feature values. It is also the width of the memory word where a non-leaf node in the discretizer is stored.
- *width of discretized feature value*: this is the width of the memory word where a leaf node in the discretizer is stored. It also determines the height of the complete binary tree.

The parameters for the C4.5 classifier are

- *number of levels in the tree:* this determines the number of stages in the architecture. The classifier built from C4.5 algorithm using the feature set we identify has 43 levels.
- *number of nodes in each level:* this determines the number of memory words needed in each tree level. This number varies from 1 to 5 in the classifier we build from Section 4.

We evaluate the impact of these parameters on the throughput and resource consumption. In our experiments on discretizer, *width of raw feature value* varies from 16 to 48 bits and *width of discretized value* from 4 to 8 bits. We vary the total number of nodes in the C4.5 classifier from 32 to 1024 and the number of levels from 6 to 48. Also, we assume a bottleneck level having 25% of the total number of nodes in the classifier. The remaining nodes are equally distributed among other levels of the tree. In addition, we evaluate the scalability of the low cost design by replicating it to support multiple input traffic streams. We also compare the resource efficiency of the high throughput design and the low cost design.

Table 5: Performance of Discretizer

High Throughput Design				Low Cost Design			
Clock Rate (MHz)				Clock Rate (MHz)			
	4 bits	6 bits	8 bits		4 bits	6 bits	8 bits
48 bits	396	402	235	48 bits	343	372	376
32 bits	440	410	252	32 bits	357	405	305
16 bits	611	593	323	16 bits	378	409	410
# of Occupied Slices				# of Occupied Slices			
	4 bits	6 bits	8 bits		4 bits	6 bits	8 bits
48 bits	191	301	379	48 bits	83	102	152
32 bits	121	186	256	32 bits	59	74	109
16 bits	70	108	148	16 bits	31	44	64
Resource efficiency (slices/Gbps)				Resource efficiency (slices/Gbps)			
	4 bits	6 bits	8 bits		4 bits	6 bits	8 bits
48 bits	0.19	0.29	0.63	48 bits	1.09	1.69	2.41
32 bits	0.11	0.18	0.40	32 bits	0.75	1.13	2.16
16 bits	0.04	0.07	0.18	16 bits	0.37	0.67	0.93

6.1 Performance of proposed C4.5 classifier

In this experiment, we evaluate the performance of our 43-level classifier implementation using our two architecture designs. This classifier is generated by running C4.5 Algorithm on our dataset using EOFS identified in Section 3.2.3. Each level of this C4.5 classifier has no more than 6 nodes.

Our high throughput design achieves a clock rate of 215 MHz. In each clock cycle two flows are classified since the distributed RAM is dual ported. As we classify the flows using the first 4 data packets, assuming 40 bytes packet size, the throughput is 550 Gbps. We observe that number of occupied slices is 6429.

The low cost design achieves a clock rate of 308 MHz. It processes two input traffic streams concurrently since block RAM is dual ported. In the worst case each input feature vector traverses through the 43 tree levels. In our architecture, operation in each tree level takes 3 clock cycles. Therefore 129 cycles are needed to classify a flow in the worst case. Assuming 4 packets per flow and 40 bytes packet size, the throughput will be 6 Gbps. The number of occupied slices is 135 in addition to the block RAM.

6.2 Scalability of discretizer / C4.5 traffic classifier

The clock rate and resource usage of the discretizer using both the designs are shown in Table 5. The row index represents width of raw feature values while the column index represents the width of discretized feature values.

The results show that both high throughput design and low cost design sustain high clock rate. However, since low cost design takes multiple cycles to discretize a value, its throughput is lower than the high throughput design. The low cost design consumes much fewer slices than high throughput design since it has only one processing element and one memory block. In our results, cost of block RAM is converted to number of slices by the calculation described in Section 5.1. We will discuss resource efficiency of both the designs in Section 6.4.

Table 7 shows the performance of high throughput design w.r.t. the total number of nodes and number of levels in the classifier. It achieves a throughput of 460.8 Gbps when realizing a 48-level classifier with 1024 nodes and a throughput of 947.2 Gbps when realizing a 6-level classifier with 32 nodes.

Table 6: Performance of Low Cost C4.5 Classifier

Total # of Nodes	# of Tree Levels	Max # of nodes per level	Clock Rate (MHz)	Throughput (Gbps)	# of Occupied Slices	Resource efficiency (slices/Gbps)
1024	48	256	334	5.94	125	21.01
512	24	128	321	11.41	117	10.22
256	24	64	308	10.95	136	12.38
128	12	32	358	25.46	121	4.77
64	12	16	338	24.04	117	4.87
32	6	8	340	48.36	125	2.59

Table 6 shows the performance of low cost design. It achieves 5.94 Gbps when realizing a 48-level classifier with 1024 nodes. We observe that the low cost design sustains a high clock rate even when we scale to large number of tree nodes. Since the number of clock cycles is linearly proportional to the number of levels in the tree to classify a flow, the throughput will drop when the number of levels of the tree increases. On the other hand, the resource usage of low cost design remains nearly the same for tree structures of different sizes.

Our low cost design can be easily replicated to support multiple input traffic streams. Its performance when processing multiple streams concurrently is shown in Table 8. High clock rate is maintained as the number of threads grow from 4 to 160. The resource usage increases linearly when the number of threads increases. Yet the resource efficiency remains the same. The results in Table 6 and Table 8 show that the low cost design is highly scalable w.r.t size of the classifier and number of input traffic streams.

Table 7: Performance of High Throughput C4.5 Classifier

Total # of Nodes	# of Tree Levels	Max # of nodes per level	Clock Rate (MHz)	Throughput (Gbps)	# of Occupied Slices	Resource efficiency (slices/Gbps)
1024	48	256	180	460.8	12429	26.93
512	24	128	206	527.36	6916	13.15
256	24	64	234	599.04	4360	7.28
128	12	32	260	665.6	2266	3.41
64	12	16	342	875.52	1328	1.52
32	6	8	370	947.2	673	0.71

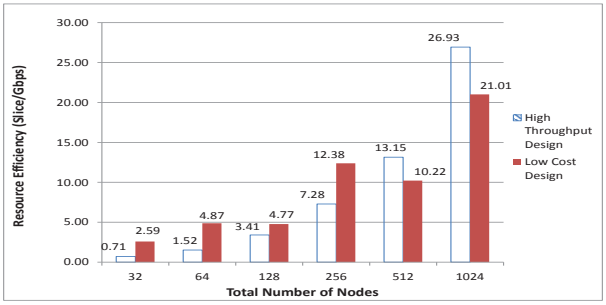
6.3 Comparison between Two Architectures

To compare the two designs, we use resource efficiency defined in Section 5.1 as the metric. The comparison between the resource efficiency of both the designs is shown in Fig-

Table 8: Performance of Multi-Stream Low Cost C4.5 Classifier

Total # of Threads	# of Nodes	# of Tree Levels	Clock Rate (MHz)	Throughput (Gbps)	# of Occupied Slices	Resource efficiency (slice/Gbps)
4	1024	48	329	11.70	330	28.21
8	1024	48	326	23.24	619	26.61
16	1024	48	348	49.50	1240	25.03
32	1024	48	330	94.03	2465	26.22
64	1024	48	330	187.75	4741	25.25
160	1024	48	315	449	11708	26.06

Figure 10: Comparison of resource efficiency of high throughput and low cost designs



ure 10. We can observe that when the size of the classifier is small, the resource efficiency of high throughput design is better than the low cost design. The same trend can be observed in Table 5. Since the discretizer is small, the high throughput design has better resource efficiency than the low cost design. However the resource efficiency of low cost design increases at a slower rate than the high throughput design. From the 32 node classifier to the 1024 node classifier, the resource usage of the low cost design stays almost the same, while the resource usage high throughput design increases by nearly 19 times. Therefore when the classifier is large, the low cost design outperforms the high throughput design in terms of resource efficiency. As shown in Figure 10 when the classifier has 1024 nodes, the resource efficiency of low cost design is 1.22 times better than the high throughput design.

7. CONCLUSION

In this paper, we proposed a high throughput architecture and a low cost architecture to realize complete online flow-level traffic classifier. We constructed a C4.5 classifier based on an emperically optimized feature set we identify. The classifier model achieved 97.92% overall accuracy in classifying 8 application protocols including classic, P2P, and instant messaging applications. Our high throughput architecture accelerated this classifier to 550 Gbps and our low cost architecture accelerated it to 6 Gbps. Both architectures were programmable to support any binary-tree-based classifier. To the best of our knowledge, ours is the first work which achieves high accuracy and high throughput at the same time.We also evaluated the performance of the two architectures w.r.t. the resource requirement of the classifier. We observed that our high throughput design outperformed the low cost design in both throughput and resource efficiency when realizing small-scale classifiers. On the other hand, the resource efficiency of our low cost architecture remained the same regardless of the number of input traffic streams. Thus it could be replicated to offer comparable throughput as our high throughput design with constant resource efficiency. This result contradicted with the common opinion that distributed RAM based design was always favored over block RAM design. In the future, we will study the tradeoffs between performance and cost of the two architectures. We will also explore the potential of our work in a broader scope of high speed classification problems.

8. REFERENCES

[1] R. Alshammari and A. N. Zincir-Heywood. Machine learning based encrypted traffic classification: identifying ssh and skype. In *the proc. of CISDA*, pages 289–296, 2009.

[2] D. Angevine and A. Zincir-Heywood. A preliminary investigation of skype traffic classification using a minimalist feature set. In *the proc. of ARES*, pages 1075 –1079, 2008.

[3] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36:23–26, April 2006.

[4] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli. Revealing skype traffic: when randomness plays with you. In *the proc. of SIGCOMM*, pages 37–48, 2007.

[5] J. Erman, M. Arlitt, and A. Mahanti. Traffic classification using clustering algorithms. In *the proc. of MineNet*, pages 281–286, 2006.

[6] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous valued attributes for classification learning. In *the proc. of IJCAI*, pages 1022–1027, 1993.

[7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.

[8] M.-H. Ho, Y.-Q. Ai, T. C.-P. Chau, S. C. L. Yuen, C.-S. Choy, P. H. W. Leong, and K.-P. Pun. Architecture and design flow for a highly efficient structured asic. *VLSI*, PP(99):1, 2012.

[9] W. Jiang and M. Gokhale. Real-time classification of multimedia traffic using fpga. In *the proc. of FPL*, 2010.

[10] T. Karagiannis, A. Broido, M. Faloutsos, and K. Claffy. Transport layer identification of p2p traffic. In *the proc. of IMC*, 2004.

[11] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. Blinc: multilevel traffic classification in the dark. In *the proc. of SIGCOMM*, 2005.

[12] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *the proc. of IJCAI*, pages 1137–1143. Morgan Kaufmann, 1995.

[13] 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 *the Proc. of ACM Co-NEXT*, '10, pages 9:1–9:12, 2010.

[14] Y. Luo, K. Xiang, and S. Li. Acceleration of decision tree searching for ip traffic classification. In *the proc. of ANCS*, 2008.

[15] A. Monemi, R. Zarei, M. Marsono, and M. Khalil-Hani. Parameterizable decision tree classifier on netfpga. *Advances in Intelligent Systems and Computing*, 182:119–128, 2013.

[16] A. W. Moore and K. Papagiannaki. Toward the accurate identification of network applications. *LNCS*, 3431/2005:41–54, 2005.

[17] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

[18] S. Sen, O. Spatscheck, and D. Wang. Accurate, scalable innetwork identification of p2p traffic using application signatures. In *the proc. of WWW*, 2004.

[19] T. Traces. http://tstat.tlc.polito.it/traces.shtml.

[20] N. Williams, S. Zander, and G. Armitage. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *SIGCOMM Comput. Commun. Rev.*, 36(5):5–16, 2006.

[21] S. Zander, T. Nguyen, and G. Armitage. Automated traffic classification and application identification using machine learning. In *the proc. of LCN*, pages 250 – 257, 2005.