

# Real-Time Classification of Multimedia Traffic using FPGA

Weirong Jiang

Ming Hsieh Department of Electrical Engineering  
University of Southern California  
Los Angeles, CA 90089, USA  
Email: weirongj@usc.edu

Maya Gokhale

Center for Applied Scientific Computing  
Lawrence Livermore National Laboratory  
Livermore, CA 94551, USA  
Email: gokhale2@llnl.gov

**Abstract**—Real-time classification of Internet traffic according to application types is vital for network management and surveillance. Identifying emerging applications based on well-known port numbers is no longer reliable. While deep packet inspection (DPI) solutions can be accurate, they require constant updates of signatures and become infeasible for encrypted payload especially in multimedia applications (e.g. Skype). Statistical approaches based on machine learning have thus been considered more promising and robust to encryption, privacy, protocol obfuscation, etc. However, the computation complexity of traffic classification using those statistical solutions is high, which prevents them being deployed in systems that need to manage Internet traffic in real time. This paper proposes a FPGA-based parallel architecture to accelerate the statistical identification of multimedia applications while maintaining high classification accuracy. Specifically, we base our design on the  $k$ -Nearest Neighbors ( $k$ -NN) algorithm which has been shown to be one of the most accurate machine learning algorithms for Internet traffic classification. To enable high-rate data streaming for real-time classification, we adopt the locality sensitive hashing (LSH) for approximate  $k$ -NN. The LSH scheme is carefully designed to achieve high accuracy while being efficient for implementation on FPGA. Processing components in the architecture are optimized to realize high throughput. Extensive experiments and FPGA implementation results show that our design can achieve high accuracy above 99% for classifying three main categories of multimedia applications from Internet traffic while sustaining 80 Gbps throughput for minimum size (40 bytes) packets.

## I. INTRODUCTION

The evolution of the Internet has induced various multimedia applications (such as Skype, IPTV, etc.) to emerge and gain rapidly popularity [1]–[3]. It becomes a vital issue for companies and Internet Service Providers (ISPs) to identify the application types of traffic carried on their networks [3], [4]. Traditional traffic classification based on well-known transport layer port numbers becomes less reliable, as emerging applications tend to hide their identify by using unpredictable port numbers [2]–[5]. Current methods to classify traffic include (1) *deep packet inspection* (DPI) which searches the packet content for the known signatures, (2) *host behavior* based schemes to exploit the connection pattern of hosts, or (3) *machine learning* based statistical methods using packet-level or flow-level features such as packet size, flow duration, etc [6]. Although DPI-based solutions can provide high accuracy, they require constant updates of signatures, fail against en-

rypted packets, and cause privacy and legal concerns. The effectiveness of host behavior based approaches depend on topological locations and traffic mixes. They are highly stateful and are inaccurate in identifying the application type for single packets [5]. Recent research on traffic classification tries to identify network applications by learning statistical patterns in externally observable features of packets or flows [1]. Such statistical approaches based on machine learning have thus been considered more promising and robust to encryption, privacy, protocol obfuscation, etc [5]–[10]. However, the practicality of the machine learning -based approaches can be hampered due to their high computation complexity and memory requirement [5], [9]. As the volume of the Internet traffic grows explosively, a large portion of which is due to multimedia applications [3], efficient hardware implementation is demanded for real-time traffic classification at high throughput beyond 10 Gbps [9].

Due to its ability to reconfigure and to offer massive parallelism, FPGA technology has long been an attractive option for implementing various real-time network processing engines [11], [12]. State-of-the-art FPGA devices such as Xilinx Virtex-6 [13] and Altera Stratix-IV [14] provide high clock rate, low power consumption, rich resources and large amounts of on-chip dual-port memory with configurable word width. This paper aims to exploit current FPGAs to achieve following goals for multimedia traffic classification:

1) *High classification accuracy*: Machine learning based approaches have been shown to be accurate and robust for classifying many applications [5]–[10]. However, little work has been done in applying these approaches to identify *multimedia* applications. It remains unclear how effective these approaches are for multimedia traffic classification.

2) *Stateless classification*: A number of existing statistical approaches exploit not only packet-level features (e.g. packet size) but also flow-level features (e.g. flow duration). While it may help improve the classification accuracy, using flow-level features requires maintaining the states of all flows and makes it hard to perform real-time classification on each packet. This paper focuses only on packet-level features which have been shown to be sufficient to achieve high accuracy [5].

3) *High throughput*: Multimedia applications are sensitive to delay and packet loss [1]. This requires the traffic classifier process packets at high throughput. For example, the current

link rate has been pushed beyond 40 Gbps, which requires classifying a packet every 8 ns in the worst case (where the packets are of minimum size i.e. 40 bytes) [12].

4) *Dynamic update*: To adapt the traffic classifier to emerging applications, new training sets may be added during the run time. The traffic classifier must support on-the-fly updates without severe performance degradation. Hence memory-based architectures are preferred to purely logic-based engines which need fully reconfiguration after each update. Moreover, the time spent on training should be minimized.

The computation kernel of a statistical traffic classifier is the machine learning algorithm. Although various accelerators including FPGA-based engines have been proposed for machine learning algorithms [15]–[20], few of them are targeted for network applications. To the best of our knowledge, none of these accelerators can meet all the performance goals stated above. In this paper, we base our design on the  $k$ -Nearest Neighbors ( $k$ -NN) algorithm which assigns to a test instance the majority class type of its  $k$  nearest neighbors. As a widely used nonparametric classification technique,  $k$ -NN has several attractive features for real-time traffic classification on FPGA. First,  $k$ -NN has been shown to be one of the most accurate statistical methods for Internet traffic classification [5]. Second,  $k$ -NN needs no or little training, which indicates low update cost. Third, unlike many other machine learning algorithms,  $k$ -NN coupled with Hamming distance requires neither multiplication nor floating point operations. Thus  $k$ -NN is well suited to be implemented on FPGA.

Given  $N$  training samples, brute-force  $k$ -NN requires  $\Omega(N)$  time to classify each test instance. Even with existing  $k$ -NN accelerators [17]–[20] which achieve up to  $P \times$  speedup by using  $P$  parallel processing elements, the performance of these accelerators for brute-force  $k$ -NN is still far from sufficient for real-time traffic classification where the number of training samples (i.e.  $N$ ) can easily exceed 100K.

We propose using locality sensitive hashing (LSH) [21] for high-performance multimedia traffic classification. LSH is an efficient data structure for *approximate* nearest neighbor problems. Compared with brute-force  $k$ -NN, the time complexity of LSH to classify a test instance is dramatically reduced to  $O(H)$  where  $H$  is the number of hash tables. Though LSH cannot guarantee finding the nearest neighbors for each test instance, we show in this paper that the classification accuracy achieved by LSH is almost as high as by brute-force  $k$ -NN. This paper makes following contributions.

- We conduct an in-depth study of using  $k$ -NN for multimedia traffic classification. After examining different distance metrics, we find that Hamming distance leads to higher accuracy than other distance metrics. This also encourages us to explore LSH which is naturally suitable being coupled with Hamming distance.
- We present the LSH-based scheme as an alternative to brute-force  $k$ -NN for traffic classification. As far as we know, our work is the first attempt to utilize LSH for high performance traffic classification. We discuss several design issues and evaluate various performance trade-offs.

With appropriate settings, our LSH scheme achieves high accuracy while being fit for implementation on FPGA.

- We detail the hardware architecture for the LSH-based traffic classifier. The main processing components in the architecture are optimized to realize high throughput. For example, we build the sorting module as a pipelined bitonic sorting network which can sort up an arbitrary sequence of inputs every clock cycle.
- Extensive simulation and implementation results on a state-of-the-art FPGA show that our design can achieve over 99% classification accuracy while sustaining 80 Gbps throughput for minimum size (40 bytes) packets. To the best of our knowledge, our work is the first FPGA design for 40+ Gbps statistical traffic classification.

The rest of the paper is organized as follows. Section II describes our methodology. Section III discusses the LSH-based traffic classification algorithms. Section IV presents the hardware architecture. Section V evaluates the performance of our schemes. Section VI reviews briefly the related work. Section VII concludes the paper.

## II. METHODOLOGY

### A. Performance Metrics

Distinct from most of previous work, this paper considers not only the accuracy but also the speed of traffic classification.

- *Accuracy* is the fraction of packets correctly classified for an application. The *overall accuracy* is the fraction of correctly classified packets over all applications.
- *Throughput* is measured as the number of packets classified per second (PPS). By multiplying it with 40 bytes i.e. the minimum size of an IP packet, worst-case raw throughput (Gbps) can be obtained.

### B. Data Set

In this paper, we focus on three main categories of multimedia applications<sup>1</sup>: VoIP, Instant Messaging (denoted as IM), IPTV. These multimedia applications have gained widely popularity and attracted great attention from both academia and industry [2], [3]. We aim to distinguish them not only between each other but also out from those legacy applications including WWW, Email, DNS, DHCP, etc.

We obtained the labeled traffic traces for the three multimedia applications: VoIP (Skype), IM (MSN, Yahoo, Jabber, Gtalk), and IPTV, from the same site [22]. As in other work [9], [10], [23], we infer the legacy applications based on well-known port numbers using Coral Reef [24]. While such port-based classification has been inaccurate for recent traffic traces, it is still applicable for some old traces such as WIDE-2000 [5], [25]. The detailed information of the traffic traces is summarized in Table I.

Since  $k$ -NN is a supervised machine learning algorithm, we need to set up the ground truth including both training and

<sup>1</sup>Though this paper considers only multimedia applications, we believe our solution can be used for identifying other categories of applications such as P2P traffic.

TABLE I  
TRAFFIC TRACES

Application	Traces	Start date & time	Duration	IP protocol	# packets	Data size (MB)
Skype	Skype1	2006-05-29 02:18:41	95 hour 26 min	TCP	2357997	338.5
	Skype2	2006-05-29 02:01:25	95 hour 45 min	UDP	39627543	8396.8
	Skype3	2006-05-29 02:49:20	79 hour 3 min	UDP	3049284	231.3
Instant Messaging (IM)	MSN	2006-05-29 02:01:25	95 hour 45 min	TCP	15434573	2234.3
	YMSG	2006-05-29 02:01:26	95 hour 45 min	TCP	841221	79.1
	XMPP	2006-05-29 02:01:25	95 hour 45 min	TCP	214636	34.8
IPTV	IPTV	2008-05-06 06:19:42	5 min 32 sec	UDP	13513514	18633.8
Legacy	WIDE-2000	2000-01-01 13:59:00	1 hour 35 min	TCP & UDP	2095192	1052.5

testing sets. The classical technique is 10-fold cross validation which divides the data set into 10 slices and runs 10 iterations: each iteration uses one of the 10 slices as the testing set and the remaining data for training set. The final result is the average of the 10 iterations. But since the number of samples per application is highly varying in traffic classification, this might lead the most prevalent application to bias the training phase of the classifier [4]. An alternative approach advocated in [5], [26] is to use a training set with the same number of samples per application. Like [5], [26], we split each labeled trace file into pieces. Half of each trace is randomly chosen into training set and the remaining half into testing set. We vary the size of the sampled training set while using the same, fixed size testing set. We randomly sample 100, 1K, 10K and 100K training packets per application from the training set, while we randomly sample 20K packets per application from the testing set. We call a packet from training sets as a *training sample*; A packet from testing set is called a *testing instance*.

### C. Feature Selection

As discussed in Section I, we consider only the packet-level features. Each packet has tens of observable features most of which are contained in the packet header. Feature selection will affect the accuracy of traffic classification. Kim et al. [5] show that it is sufficient to achieve high accuracy by using IP protocol, packet size, TCP/UDP ports, TCP flags as the features. We adopt all these suggested features but TCP flags, since some applications to be identified in our work are based on UDP whose packets do not contain TCP flags.

## III. LSH-BASED TRAFFIC CLASSIFICATION

As discussed in Section I,  $k$ -NN is among the most accurate algorithms for Internet traffic classification [5]. But its brute-force implementation, which can find the *exact*  $k$  nearest neighbors for a test instance, cannot meet the performance requirements for real-time traffic classification. On the other hand, LSH was proposed for solving the *approximate*  $k$ -NN problems [21]. This section discusses using these two  $k$ -NN algorithms for accurate classification of *multimedia* traffic. Some design issues such as the selection of distance metrics are also addressed.

### A. Baseline: Brute-Force $k$ -NN

We first study the performance of using brute-force  $k$ -NN for multimedia traffic classification. This will be the baseline to which we compare our LSH-based algorithm.

Let  $R$  be the training set,  $r$  any training sample and  $q$  the test instance. The brute-force  $k$ -NN algorithm is:

- 1: Initialize a  $k$ -entry list  $L = \emptyset$  to maintain the  $k$  training samples which have the smallest distances to  $q$ .
- 2: **for all**  $r \in R$  **do**
- 3:   Compute the distance from  $q$  to  $r$ .
- 4:   Insert  $r$  into  $L$  while keeping the  $k$  entries in  $L$  to be sorted in ascending order of their distances to  $q$ .
- 5: **end for**
- 6: Assign to  $q$  the application type that is the majority among the  $k$  training samples in  $L$ .

When the training set  $R$  contains  $N$  training samples, the brute-force  $k$ -NN algorithm takes  $\Theta(Nk)$  time to classify a test instance. An important issue for  $k$ -NN based classification is the selection of distance metrics. While  $k$ -NN can be coupled with various distance metrics, most of previous work considers only Euclidean distance. Recall that computing Euclidean distance requires multiplication, which may not be efficient to be implemented on FPGA compared with other distance metrics such as Hamming distance. We need to examine the impact of different distance metrics on the accuracy of  $k$ -NN -based classification. Table II shows the experimental results including the classification accuracy and the average classification time per test instance. In these experiments, we set  $k = 1$  and vary the number of training samples per application. Three distance metrics are considered: Euclidean, Manhattan and Hamming distances. The experiments are performed on a Intel P4/Xeon 2.26 GHz dual processor with 8K L1, 512K L2, and 1024K L3 cache, 3GB memory.

TABLE II  
CLASSIFICATION PERFORMANCE OF BRUTE-FORCE  $k$ -NN USING DIFFERENT DISTANCE METRICS

# of Training samples	Overall accuracy			Classification time per test instance
	Euclidean	Manhattan	Hamming	
100	84.40%	85.37%	85.76%	2.29 msec
1K	86.22%	86.51%	87.30%	16.16 msec
10K	99.73%	99.83%	99.91%	147.75 msec
100K	100%	100%	100%	1431.75 msec

As we expected, a larger number of training samples result in higher classification accuracy and higher computation time. The classification time for each test instance is proportional to the number of training samples. Another notable observation is that Hamming distance outperforms other two distance metrics with respect to the classification accuracy. This encourages us to exploit LSH, which is discussed in the following section.

## B. Locality Sensitive Hashing

Table II shows that it takes more than 1 second for brute-force  $k$ -NN to classify a test instance against 100K training samples. Such performance cannot achieve 10+ Gbps throughput for real-time traffic classification. To reduce the computation complexity of brute-force  $k$ -NN, LSH was proposed for searching approximate nearest neighbors [21]. Its main idea is to hash the data points using several hash functions so as to ensure that, for each function, the probability of collision is much higher for points which are close to each other than for those which are far apart. Then, one can determine near neighbors by retrieving the points that are hashed into the same bucket as the query point. LSH-based  $k$ -NN algorithm cannot guarantee finding the  $k$  nearest neighbors for a test instance.

The LSH algorithm relies on the existence of locality-sensitive hash functions. One of the easiest ways to construct a LSH function is by bit selection. A random function simply selects random bits from an input point. This approach works efficiently for Hamming distance [21]. Also as shown in Table II,  $k$ -NN coupled with Hamming distance achieves the highest classification accuracy. Hence we adopt Hamming distance and build the LSH functions by bit selection.

Let  $R$  denote the training set,  $r$  any training sample and  $q$  the test instance. We design LSH-based classification as the following two phases.

- Training (Preprocessing)
  - 1: Choose  $H$  LSH functions:  $g_i, i = 1, 2, \dots, H$ .
  - 2: Construct  $H$  hash tables.
  - 3: **for all**  $r \in R$  **do**
  - 4:   **for**  $i = 1$  to  $H$  **do**
  - 5:     Map  $r$  hashed using  $g_i$  onto the  $i$ th hash table.
  - 6:   **end for**
  - 7: **end for**
- Testing (Querying)
  - 1: Initialize a  $k$ -entry list  $L = \emptyset$  to maintain the  $k$  training samples having the smallest distances to  $q$ .
  - 2: **for**  $i = 1$  to  $H$  **do**
  - 3:   Retrieve the training sample from the bucket  $g_i(q)$  in the  $i$ th hash table.
  - 4:   Compute the distance from the retrieved training sample to  $q$ .
  - 5:   Insert  $r$  into  $L$  while sorting the  $k$  entries in  $L$  in ascending order of their distances to  $q$ .
  - 6: **end for**
  - 7: Assign to  $q$  the application type that is the majority among the  $k$  training samples in  $L$ .

The LSH-based classification using  $H$  hash functions takes  $\Theta(NH)$  time for training  $N$  samples and  $\Theta(Hk)$  time for classifying each test instance. We conduct experiments with varying the number of training samples. Table III shows the performance including classification accuracy and computation time for both training and testing. In these experiments, we initiate one LSH-based classifier for TCP and UDP applications, respectively. We use a total of 16 hash tables with different LSH functions. Each hash table has 1K buckets. We set  $k = 2$ .

TABLE III  
CLASSIFICATION PERFORMANCE OF LSH-BASED ALGORITHM

# of Training samples	Overall accuracy	Training time	Classification time per test instance
100	81.24%	0 sec	0.023625 msec
1K	87.24%	0.05 sec	0.024625 msec
10K	99.79%	0.53 sec	0.025500 msec
100K	99.97%	5.33 sec	0.025750 msec

Comparing Table III with Table II, our LSH-based algorithm achieves the similar classification accuracy as brute-force  $k$ -NN while the classification time per test instance is dramatically reduced. In case of 100K training samples, the classification time is almost 5 orders of magnitude lower than that of brute-force  $k$ -NN. However, the corresponding throughput of LSH-based software implementation is nearly 40 thousand packets per clock cycle (KPPS), which is still far from meeting real-time requirement.

## IV. ARCHITECTURE

The proposed LSH-based classification algorithm is highly desirable for hardware implementation. We design the hardware architecture shown in Figure 1. There are two LSH-based classifiers: one for TCP and the other for UDP applications. Each LSH-based classifier consists of  $H$  hash tables,  $H$  distance calculators, a sorting module and a majority voter. The size (i.e. the number of buckets) of each hash table is  $M$ . These architectural parameters, i.e.  $k$ ,  $H$  and  $M$ , can be configured based on performance requirements (see Section V-A for details).

### A. Processing Flow

For either training or testing, we extract from an input packet the selected features including IP protocol, packet size and port numbers. According to its IP protocol (TCP or UDP), the packet is directed to the corresponding LSH-based classifier. In other words, TCP (UDP) packets will go through the LSH-based classifier for TCP (UDP).

During the training phase, each training sample is mapped to the  $H$  hash tables in parallel. Each entry in a hash table contains the content of a training sample, i.e. all the selected features and the application type.

During the testing phase, each test instance is hashed into the hash keys to retrieve one training sample from each hash table. Then the distance is computed between the test instance and each retrieved training sample. The obtained  $H$  distance values are sorted by the sorting module into ascending order. The final output is the majority application type among the  $k$  training samples that have the smallest distance values.

### B. Optimizing Components

To achieve high throughput for real-time traffic classification, we pipeline the architecture by inserting registers between each processing component. We also optimize main processing components in the architecture by exploiting the features provided by current FPGAs.

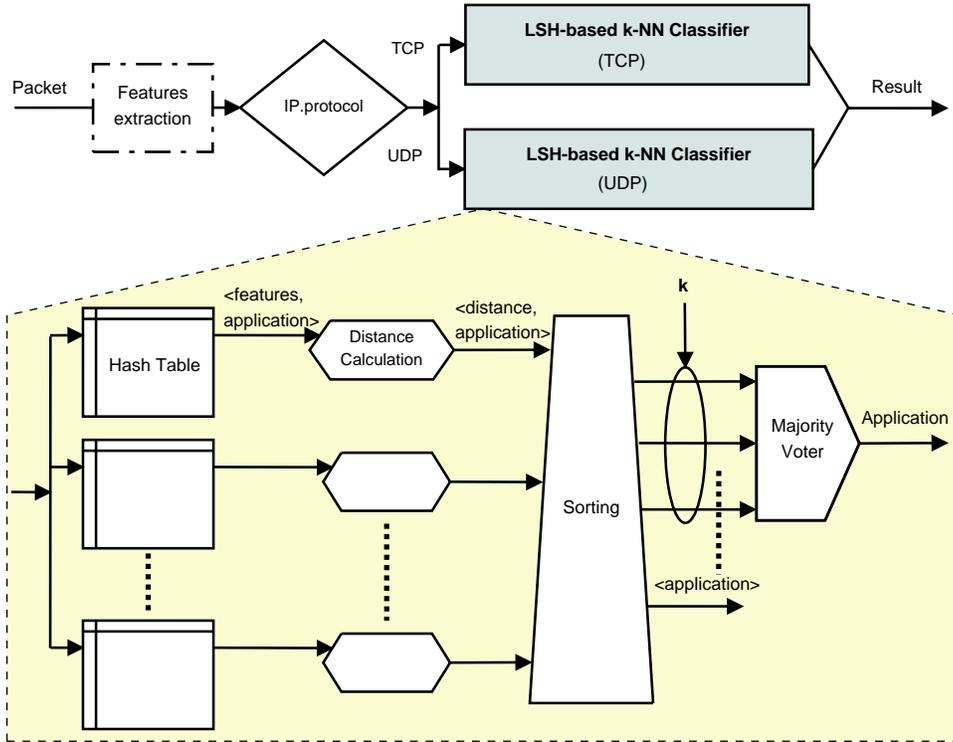


Fig. 1. Block diagram of the LSH-based  $k$ -NN classification architecture.

1) *Hash Tables*: The  $H$  hash tables employ different LSH functions. Each hash table consists of the logic to generate the hash key and a memory block to store the training samples. We utilize the dual-port RAMs available in current FPGAs to allow processing two packets in the same clock cycle.

2) *Sorting*: We implement the sorting module as a parallel bitonic sorting network [27] which can sort up  $n$  elements of arbitrary order in  $\frac{\log n(1+\log n)}{2}$  clock cycles. We pipeline the bitonic sorting network into  $\frac{\log n(1+\log n)}{2}$  stages to achieve throughput of one sorted output per clock cycle.

With these optimizations, our architecture can process two packets every clock cycle. The total number of clock cycles for a packet to go through the LSH-based classification architecture with  $H$  hash tables is  $\frac{\log H(1+\log H)}{2} + 3$ , where  $\frac{\log H(1+\log H)}{2}$  clock cycles are spent on sorting the  $H$  outputs from hash tables, one clock cycle on hash table access, one clock cycle on Hamming distance calculation, and one clock cycle on majority voter.

### C. Dynamic Update

Our architecture can easily support dynamic updates for adding new training samples. As both training and testing packets go through the same datapath, we append one single bit for each input packet to distinguish whether it is a training or testing packet. This single bit can also be used as a write enable signal for writing the training sample into the hash table. Since our architecture is linear, any testing packet preceding or following a training packet can perform its operations while the training packet performs an update.

Though dual-port RAMs are used, we allow only one write per clock cycle for each RAM to avoid write conflicts. Thus the sustained throughput for updates (i.e. training) is one packet per clock cycle, i.e. half the classification throughput.

## V. IMPLEMENTATION AND EVALUATION

We implement the LSH-based classification architecture as a parameterized design on FPGA. The target device is Xilinx Virtex 5 xc5v1x50t with -3 speed grade. All the following implementation results are from Xilinx ISE 11.1 post place and route reports. The main architectural parameters include  $k$ , the number of hash tables ( $H$ ) and the hash table size ( $M$ ). We examine the performance trade-offs between the classification accuracy and the resource utilization, by varying those parameters. In the following experiments, the default settings are  $k = 2$ ,  $H = 16$ ,  $M = 1K$ . The classification accuracy results are obtained based on 100K training samples.

### A. Performance Tradeoffs

1)  $k$ : We conducted experiments by varying  $k$ . Table IV shows that  $k$  did not have big impact on the performance.

TABLE IV  
IMPACT OF  $k$

k	Overall accuracy	Slice usage	BRAM usage
1	99.93%	28%	40%
2	99.97%	27%	40%
3	99.86%	30%	40%
4	99.90%	27%	40%

2) *Number of Hash Tables*: We varied the number of hash tables:  $H = 2, 4, 8, 12, 16$ . Table V shows the results including overall accuracy and resource utilization. As we expected, using more hash tables resulted in higher classification accuracy and higher resource utilization.

TABLE V  
IMPACT OF THE NUMBER OF HASH TABLES

# of Hash tables	Overall accuracy	Slice usage	BRAM usage
2	73.58%	1%	3%
4	97.56%	5%	9%
8	99.56%	13%	20%
12	99.83%	22%	30%
16	99.97%	27%	40%

3) *Hash Table Size*: We varied the hash table size:  $M = 256, 512, 1K, 2K$ . Table VI shows the results including overall accuracy and resource utilization. Using larger hash tables resulted in higher classification accuracy. But we did not observe the expected increase in BRAM usage when the hash table size was increased before reaching 2K. This was due to that the FPGA implementation did not utilize Block RAMs efficiently when the hash table size was smaller than 1K.

TABLE VI  
IMPACT OF HASH TABLE SIZE

Hash table size	Overall accuracy	Slice usage	BRAM usage
256	99.34%	28%	40%
512	99.83%	29%	40%
1K	99.97%	27%	40%
2K	99.98%	29%	80%

### B. Performance Summary

With the default settings ( $k = 2, H = 16, M = 1K$ ), our design met the timing constraints to achieve 125 MHz clock rate. Since dual-port RAMs are used, the sustained throughput is 250 million packets per second, i.e. 80 Gbps for minimum size (40 bytes) packets. The overall resource utilization is shown in Table VII.

TABLE VII  
RESOURCE UTILIZATION

	Used	Available	Utilization
Number of Slices	1,998	7,200	27%
Number of bonded IOBs	120	480	25%
Number of Block RAMs	26	60	43%
Total Memory used (Kb)	864	2,160	40%

Using 100K training samples, we obtain the detailed classification accuracy for each application, as shown in the confusion matrix in Table VIII. The rows represent the application types of the testing set while the columns represent the classification results. We can see that our design achieves high classification accuracy for these multimedia applications.

Table IX compares the performance of the schemes discussed in this paper. The results are based on 100K training samples. SW stands for software implementation while HW

TABLE VIII  
CONFUSION MATRIX FOR LSH-BASED CLASSIFICATION

	Skype	IM	IPTV	Legacy	Unknown
Skype	<b>99.865%</b>	0%	00.005%	00.13%	0%
IM	0%	<b>100%</b>	0%	0%	0%
IPTV	0%	0%	<b>100%</b>	0%	0%
Legacy	0%	0%	0%	<b>100%</b>	0%

the hardware design. The FPGA implementation of LSH-based classifier achieves 3 and 8 orders of magnitude higher throughput than the software implementations of LSH and brute-force  $k$ -NN -based classification schemes, respectively.

TABLE IX  
COMPARISON OF THE SCHEMES DISCUSSED IN THIS PAPER

	Brute-force $k$ -NN (SW)	LSH (SW)	LSH (HW)
Overall accuracy	100%	99.97%	99.97%
Throughput	0.7 PPS	40 KPPS	250 MPPS
Latency	1.5 sec	26 $\mu$ s	104 ns

## VI. RELATED WORK

Substantial attention has been attracted in exploiting machine learning algorithms for traffic classification. A comprehensive survey is provided in [7]. However, most of previous work in this area focuses on the accuracy and robustness of classification. The only work we found on accelerating machine learning -based traffic classification is [28] by Luo et al., which aims at C4.5 decision tree. Their objective is to reduce the number of memory accesses for classifying each packet by building a shorter and fatter tree. The tree height is reduced at the cost of using more memory banks. At some internal node, a varying number of memory accesses may be needed for branching decision. Since the authors have not implemented their design on FPGA, the actual performance results are unclear. Note that the number of memory accesses for each packet to traverse the decision tree is varying and highly depends on the data set. This makes it difficult to pipeline the tree traversal to achieve high throughput. Canini et al. [1] use flow-level features from the first several packets per flow and employ C4.5 algorithm for machine learning. They implement the machine learning module in software and the flow lookup table in hardware. The maximum throughput of their NetFPGA implementation is 8 MPPS.

In addition to LSH, many algorithms have been proposed to improve the performance of  $k$ -NN. The techniques called  $k$ -NN condensation or  $k$ -NN thinning, have been studied over years to reduce the size of the training set without lowering the accuracy of  $k$ -NN [29].  $k$ -NN thinning algorithms are also computation-intensive and have attracted recent efforts for FPGA acceleration [30]. The  $k$ -d tree algorithm [31] organizes the training data into a binary tree where each training sample is stored as a node. While  $k$ -d tree improves performance by decomposing the search space, it relies on complex data structure and recursive tree traversal algorithms, neither of which maps well onto FPGA.

## VII. CONCLUSION

This paper proposes a FPGA-based parallel architecture for real-time multimedia traffic classification. We base our design on  $k$ -NN -based classification algorithm and propose using locality sensitive hashing (LSH) to improve classification speed. Our FPGA implementation of the LSH-based traffic classifier can achieve above 99% classification accuracy for distinguishing three main categories of multimedia applications from the legacy Internet applications. By exploiting the features provided by state-of-the-art FPGAs, our design sustains 80 Gbps throughput for minimum size (40 bytes) packets, while consuming small amount of on-chip resources. Various design trade-offs are also discussed. To the best of our knowledge, this work is the first FPGA design for 40+ Gbps Internet traffic classification. Our future work includes applying the proposed solution to classifying more application types, and porting the FPGA design onto development boards and testing its performance under real-life network traffic.

## VIII. ACKNOWLEDGEMENT

This work was funded by the Lawrence Livermore National Laboratory LDRD programs Storage-Intensive Supercomputing project under DOE contract W-7405-ENG-48.

## REFERENCES

- [1] M. Canini, W. Li, M. Zadnik, and A. W. Moore, "Experience with high-speed automated application-identification for network-management," in *ANCS'09: Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ACM, Oct 2009.
- [2] J. Fan, D. Wu, A. Nucci, R. Keralapura, and L. Gao, "Protocol oblivious classification of multimedia traffic," *Security and Communication Networks*, 2009, published online in Wiley InterScience.
- [3] F. Hao, M. Kodialam, and T. Lakshman, "On-line detection of real time multimedia traffic," in *ICNP '09: Proceedings of the 17th IEEE International Conference on Network Protocols*, Oct. 2009, pp. 223–232.
- [4] M. Pietrzyk, J.-L. Costeux, G. Urvoy-Keller, and T. En-Najjary, "Challenging statistical classification for operational usage: the adsl case," in *IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009, pp. 122–135.
- [5] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: myths, caveats, and the best practices," in *CoNEXT '08: Proceedings of the 2008 ACM CoNEXT Conference*. ACM, 2008, pp. 1–12.
- [6] M. Zhang, W. John, K. Claffy, and N. Brownlee, "State of the art in traffic classification: A research review," in *PAM '09: 10th International Conference on Passive and Active Measurement, Student Workshop*. ACM, 2009, pp. 1–2.
- [7] T. Nguyen and G. Armitage, "A Survey of Techniques for Internet Traffic Classification using Machine Learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.
- [8] A. Dainotti, W. de Donato, A. Pescapé, and P. Salvo Rossi, "Classification of network traffic via packet-level hidden markov models," in *GLOBECOM '08: Proceedings of IEEE Global Telecommunications Conference 2008*, 2008, pp. 1–5.
- [9] L. Salgarelli. Statistical traffic classification in IP networks: challenges, research directions and applications. [Online]. Available: <http://netgroup.polito.it/teaching/trc/0708/Salgarelli-statisticaltrafficclassification.pdf>
- [10] 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.*, vol. 36, no. 5, pp. 5–16, 2006.
- [11] M. Zadnik, M. Canini, A. W. Moore, D. J. Miller, and W. Li, "Tracking elephant flows in internet backbone traffic with an FPGA-based cache," in *Proceedings of the 19th International Conference on Field Programmable Logic and Applications (FPL'09)*, Aug 2009.
- [12] W. Jiang and V. K. Prasanna, "Large-scale wire-speed packet classification on FPGAs," in *FPGA '09: Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2009, pp. 219–228.
- [13] Xilinx Virtex-6 FPGA Family: The High-Performance Silicon Foundation for Targeted Design Platforms. [Online]. Available: [www.xilinx.com/products/virtex6/](http://www.xilinx.com/products/virtex6/)
- [14] Altera Stratix IV FPGA: High Density, High Performance AND Low Power. [Online]. Available: <http://www.altera.com/products/devices/stratix-fpgas/stratix-iv/>
- [15] K. Irick, M. DeBole, V. Narayanan, and A. Gayasen, "A hardware efficient support vector machine architecture for FPGA," in *FCCM '08: Proceedings of the 2008 16th International Symposium on Field-Programmable Custom Computing Machines*. IEEE Computer Society, 2008, pp. 304–305.
- [16] J. Zhu and P. Sutton, "FPGA implementations of neural networks - a survey of a decade of progress," in *Proc. FPL*, 2003, pp. 1062–1066.
- [17] A. Ferrari, M. Borgatti, and R. Guerrieri, "A vlsi array processor accelerator for k-nn classification," in *ICPR '96: Proceedings of the International Conference on Pattern Recognition (ICPR '96)*, vol. 4. IEEE Computer Society, 1996, p. 723.
- [18] M. Tahir and A. Bouridane, "An FPGA based coprocessor for cancer classification using nearest neighbour classifier," in *Proceedings of 2006 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '06)*, vol. 3, may 2006, pp. III –III.
- [19] H.-Y. Li, Y.-J. Yeh, and W.-J. Hwang, "Using wavelet transform and partial distance search to implement k NN classifier on FPGA with multiple modules," in *ICIAR*, 2007, pp. 1105–1116.
- [20] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using gpu," in *Proc. CVPR Workshop on Computer Vision on GPU*, June 2008.
- [21] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, vol. 51, no. 1, pp. 117–122, 2008.
- [22] Tstat traces. [Online]. Available: <http://tstat.tlc.polito.it/traces.shtml>
- [23] A. Este, F. Gringoli, and L. Salgarelli, "On the stability of the information carried by traffic flow features at the packet level," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 3, pp. 13–18, 2009.
- [24] CoralReef software suite. [Online]. Available: <http://www.caida.org/tools/measurement/coralreef/>
- [25] WIDE daily trace of a trans-Pacific T1 line (2000). [Online]. Available: <http://mawi.wide.ad.jp/mawi/samplepoint-A/2000/>
- [26] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification," in *CoNEXT '06: Proceedings of the 2006 ACM CoNEXT conference*. ACM, 2006, pp. 1–12.
- [27] K. E. Batcher, "Sorting networks and their applications," in *AFIPS '68 (Spring): Proceedings of the April 30–May 2, 1968, spring joint computer conference*. ACM, 1968, pp. 307–314.
- [28] Y. Luo, K. Xiang, and S. Li, "Acceleration of decision tree searching for IP traffic classification," in *ANCS'08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ACM, 2008.
- [29] F. Angiulli, "Fast nearest neighbor condensation for large data sets classification," *IEEE Trans. on Knowl. and Data Eng.*, vol. 19, no. 11, pp. 1450–1464, 2007.
- [30] T. Schumacher, C. Pleschl, and M. Platzner, "An accelerator for k-th nearest neighbor thinning based on the IMORC infrastructure," in *Proceedings of the 19th International Conference on Field Programmable Logic and Applications (FPL'09)*. IEEE, Sep. 2009, pp. 338–344.
- [31] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.