# Exploring Textures in Traffic Matrices to Classify Data Center Communications

Celio Trois*, Luis C. Bona*, Luiz S. Oliveira*, Magnos Martinello†, Douglas Harewood-Gill†,
Marcos D. Del Fabro*, Reza Nejabati†, Dimitra Simeonidou†, Joao C. D. Lima‡, Benhur Stein‡
*Computer Science Dept, Federal University of Parana, Curitiba, Brazil,
Emails: {ctrois, bona, lesoliveira, didonet}@inf.ufpr.br
†High Performance Networks Group, University of Bristol, Bristol, United Kingdom,
Emails: {magnos.martinello, douglas.harewood-gill, reza.nejabati, dimitra.simeonidou}@bristol.ac.uk
‡Computer Science Dept, Federal University of Santa Maria, Santa Maria, Brazil, Emails: {caio, benhur}@inf.ufsm.br

***Abstract*—Data analytics and scientific computing are two modern applications that in recent years have substantially changed their computation and communication needs, requiring additional processing capability and bandwidth to be able to keep pace with current demands. These applications are commonly processed within data centers, exchanging enormous volumes of data, rapidly stressing existing network infrastructures. Thus, it is crucial for data center operations and management to be able to understand and classify the communication demands of these applications. The traditional approaches for classifying application traffic are port-based and Deep Packet Inspection, both presenting issues with current network technology. Some recent works propose using machine learning plus statistical information collected from application flows to classify traffic. Applications running in data centers present communication patterns which can be recognized through their traffic matrices. So, the main contribution of this paper is a method that explores the textural information extracted from these matrices to classify the data center traffic using machine learning techniques. As a proof-of-concept, we implemented this method in a system named DCTraCS. The experimental dataset was gathered from two real data centers, collecting the traffic matrices of MapReduce and a set of scientific applications every second for a period of 30 minutes. For assessing our proposal, we compared it with other machine learning techniques for classifying application traffic found in current literature. Results show that our approach achieved the highest accuracy, classifying correctly over 99% of our data center applications.**

## I. Introduction

In recent years, enterprises and universities are increasingly employing data centers and large clusters for running a variety of applications. These range from social networking and gaming to computation-intensive applications such as indexing Web content, data analysis, and scientific computing [1]. Scientists are using these applications to create and predict complex phenomenas, for example, weather forecasting, prediction of natural disasters, bacterial profiling, animal genotyping, and so forth. Applications running on data centers have also a wide range of business areas, such as, analyzing large volumes of customer data or logs from monitoring real-time network, simulating product designs, modeling complex workflows, and exploring many aspects of social networks.

An interesting aspect is that the vast majority of these applications express similar computation and communication patterns [2], meaning that they tend to transmit the same amount of data across the same computing nodes regardless of the input data. These patterns are intensely researched and were used for optimizing the communication on networks-on-chip [3], graphics processing units (GPUs) [4], multiprocessor architectures [5], and ameliorating the performance of applications [6].

For improving the quality of the results, these applications are increasingly demanding computational power, being executed in dedicated computing systems, taking many hours or even days to complete their executions, and moving huge volumes of data across the computing nodes, quickly stressing the capabilities of modern networks. So, the understanding of the communication demands and classifying the applications within data centers has become a major challenge in networking research [7]. The correct identification of which applications are using the data center resources is essential in several management activities, such as scaling and expansion planning [8], traffic engineering [9], detection of anomalies [10, 11], monitoring [12], virtual machine (VM) placement [13], and energy saving [14].

Existing proposals for classifying data center communications fall into three categories where each one has its own Achilles' heel. The first approach uses port-based classification process and has been recognized as being inaccurate as many applications adopted dynamic port numbering to overcome the performance limitations of networks. Also, to bypass the security policies imposed by firewalls, some applications use standard port numbers assigned by IANA (eg, 80-HTTP or 22-SSH) on their own communication protocols [15].

Another approach is based on Deep Packet Inspection (DPI) which consists of examining the payloads of the packets for classifying traffic. This approach not only imposes significantly higher computational complexity but also requires specific knowledge of the application protocols [16]. Furthermore, many applications are adopting cryptographic methods to ensure security in the communication between their computational nodes and therefore preventing DPI from classifying any traffic.

In a third category, Machine Learning (ML) techniques are used for traffic classification [17, 18, 19, 20]; these methods

work by exploiting intrinsic and statistical flow information for feeding the ML classifiers. For instance, packet size average and variance, total number of packets or bytes, flow duration, server and client port numbers, and many others. All of these ML methods use the same type of representation (flows statistics) and, to get out of this local minimum, it is necessary to investigate other information that allows discriminating the different types of network traffic.

Unlike existing traffic classification techniques, we propose a novel method employing characteristics (features) extracted from the applications' traffic matrices (TMs), as an input for the classifiers. A TM can be formalized as a matrix $M$, where each position $M_{[i][j]}$ holds the number of bytes transmitted during a time $t$, from node $i$ to node $j$. For classifying, we consider the application TM as an image where every pixel $(p_{ij})$ represents the amount of bytes of $M_{[i][j]}$. The image is drawn in a gray gradient, where black means most intensive communication and white means no communication.

The problem is that the data center task scheduler can randomly allocate the processes on the computing nodes, thus generating a different visual pattern for each execution. To remedy this problem, we developed a function for "unscrambling" the TMs. This function creates a visual texture for each application, regardless of the processes allocation order.

Our method was implemented in a system named DCTraCS (**D**ata **C**enter **Tra**ffic **C**lassification **S**ystem) and, for assessing it, we applied two well-known textural representations, Uniform LBP (ULBP) [21] and Robust LBP (RLBP) [22] for extracting the feature vectors. Then we used different machine learning classifiers and the best results, though, have been achieved by Gaussian Support Vector Machines (SVM) [23] and Random Forest (RF) [24].

We carry out a comparison of classification performance using our method and other ML approaches, described in current literature. Our experimental dataset has been gathered from MapReduce [25] and a set of scientific applications running on real data centers, collecting the TMs every second for a period of 30 minutes. Our approach resulted in a classification accuracy of over 99%, while the best result for the other methods was 87.6%. This finding revealed that the textural features extracted from the TMs are more discriminative and robust for classifying data center traffic.

The rest of this paper is structured as follows. Section II presents the background and related works; Section III shows the implementation details of DCTraCS; Section IV details the experiments executed in two real data centers, and finally, our conclusions are discussed in Section V.

## II. Background and Related Works

From reviewing current literature, we found several works using Machine Learning (ML) methods for traffic classification. In this section, we briefly summarize the ML process and then we present some relevant works which achieved significantly high classification accuracy.

ML is the general name for a series of different algorithms (such as Linear Regression, Decision Tree, Support Vector Machine, etc) designed to use training datasets previously collected for making predictions. It can be divided into two areas: regression for predicting continuous variables, and pattern recognition for assigning predefined class labels to particular observations, grouping them into discrete categories.

Pattern recognition tasks can be grouped into two main sub-categories: supervised and unsupervised learning. Unsupervised learning methods deal with unlabeled instances where the classes have to be inferred from the unstructured dataset. Typically, unsupervised learning employs a clustering technique to group the unlabeled samples based on certain similarity (or distance) measures. In contrast, in supervised learning, the class labels in the dataset used to build the classification model are known. Supervised learning has been used for traffic classification so, next we present some relevant works.

In 2006 Bernaille et al. [26] proposed a technique using an unsupervised ML (Simple K-Means) algorithm that classified different types of TCP-based applications using the first few packets of the traffic flow. Their experiments aimed to classify ten applications, achieving a correct identification over 80%. Eerman et al. [17] classified the traffic using supervised (Naïve Bayes classifier) and unsupervised learning (Expectation Maximization clustering algorithm). Their testbed was composed of traffic traces collected from University of Auckland, trying to classify eight applications. Their results show an accuracy of over 91%.

Soysal and Schmidt [27] investigated and evaluated the classification performance of three supervised ML algorithms (Bayesian Networks, Decision Trees, and Multilayer Perceptrons) for classifying six different types of traffic. Their datasets were acquired from the National Academic Network of Turkey, and the accuracy reported in their paper ranges from 95% to 97%. Zhang et al. [19] developed a feature selection algorithm which pre-filters most of the features and further uses a wrapper method to select the best features for a specific classifier. Their approach was evaluated using three classifiers from the traces captured from different networks, achieving more than 94% flow accuracy and 80% byte accuracy on average.

Fahad et al. [20] proposed a method for identifying both optimal and stable features relying on a multi-criterion fusion-based feature selection technique. They used traffic dataset collected from the University of Cambridge, classifying twelve applications with five classifiers (K-Nearest Neighbours, Naïve Bayes, Decision Tree, Support Vector Machine, and Logistic Regression), and getting an accuracy ranging from 70% to 97%. Apart from these works, there were some surveys published [15, 7] reporting existing techniques for traffic classification using ML.

The current state-of-the-art for traffic classification has employed ML based essentially on statistical information collected from network flows for creating feature vectors that feed the classifiers. In contrast to previous proposals, our approach considers a holistic perspective of all application flows by "taking pictures" from the traffic and using their

textural information as input for ML classifiers.

## III. Data Center Traffic Classification System

Considering that parallel applications present communication patterns, in an earlier work [28] we observed that each application has a singular TM, which could be used to distinguish it from the other applications. So, in this work, we propose using the visual textures expressed by the applications' TMs for classify them. Visual texture is strictly two-dimensional and it can be defined as something that allows identifying different things, such as plaid, stripes, a brick wall, or a piece of burlap.

To evaluate our proposal for classifying the application traffic, we developed the **D**ata **C**enter **Tra**ffic **C**lassification **S**ystem (DCTraCS)[1]. The system architecture is shown in Fig. 1 and explained in the following sections.

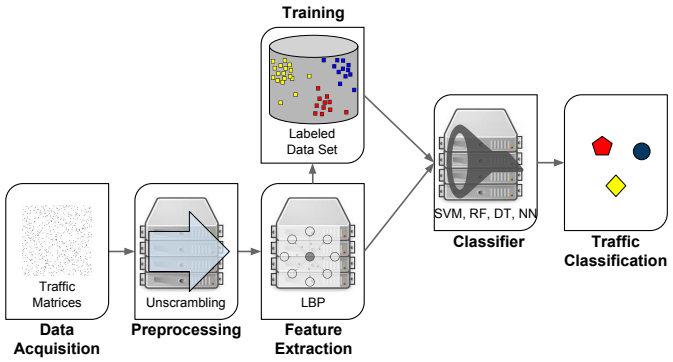


Fig. 1. DCTraCS processing overview.

The basic operation of DCTraCS is first to acquire the TMs. Next, we apply a novel function for unscrambling the TMs, rendering different visual textures for each application, regardless the order processes were allocated on computing nodes. Then, well-known visual descriptors are used for extracting the feature vectors used by the classifiers for identifying the applications.

### A. Data Acquisition: Getting the Traffic Matrices

The first step of a pattern recognition system is the data acquisition. For DCTraCS, the input depends on collecting the applications' TMs. As stated before, each TM is represented as a matrix $M_{[i][j]}$ holding the number of bytes transmitted between each pair of computing nodes. The entire execution of an application can be seen as a set of multiple TMs, collected for every time $t \rightarrow M_{[i][j]}(t)$.

The TMs are normalized to their maximum values, forming a grayscale image, where cells in black are the most communicative pair of nodes and white cells indicate that no communication happened. An important thing to note is that the manner of collecting the TMs is orthogonal to DCTraCS, which means that it might be obtained using different techniques [29, 30].

[1]DCTraCS is publicly available at: www.inf.ufpr.br/ctrois/dctracs/

### B. Preprocessing: The Unscrambling Function

Usually, the collected data for pattern recognition can not be easily processed by computer algorithms and some preprocessing must be performed for facilitating the classification. In the context of this work, when the users want to execute an application in a data center environment, they have to submit it to a job tracker or queuing system. This system is responsible for automatically allocating the required computing resources and returning a list of computing nodes ($cn_1$, $cn_2$, ..., $cn_n$). The problem happens when the processes are randomly allocated on the computing nodes, precluding the recognition of applications by the proposed method.

Fig. 2 shows TMs when the application's processes ($ap_1$, $ap_2$, ..., $ap_n$) are allocated in an ordered manner (i.e. $ap_1$ in $cn_1$, $ap_2$ in $cn_2$, ..., $ap_n$ in $cn_n$) and when the processes are randomly distributed on the computing nodes ("*Hosts in Random Order*" column). If the processes are allocated in a random order, it is not possible to identify any visual textures because the TMs are composed of many scattered points.

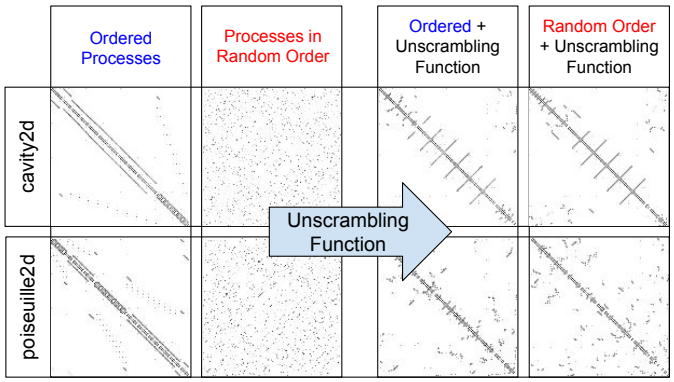


Fig. 2. The TMs of cavity2s and poiseuille2d applications for processes allocated in an ordered and in a random order.

For tackling this problem, we developed an *unscrambling function* that reorders the TM lines and columns, rendering a different texture for each application. The basic premise of this function is, for each line, "bringing" the most communicating pair of nodes close to the matrix main diagonal. As we can see, in the two right columns of Fig. 2, after applying the unscrambling function, it is possible to visually distinguish the applications looking at the generated textures, independently of the hosts allocation order.

The Algorithm 1 presents the unscrambling function. For every line in the TM, it first locates the column with the highest value (above the main diagonal), and then swaps the entire column, bringing it close to the main diagonal. As the computing nodes are the $x$ and $y$-axes of the TM, it is mandatory also to swap the line situated in the same position of the swapped column.

### C. Feature Extraction

After collecting and unscrambling the TMs, the next step consists of extracting the feature vector. As stated before, our

**Algorithm 1** The unscrambling function

```
1: function UNSCRAMBLEMATRIX(mat)
2:     max ← 0
3:     max_pos ← −1
4:     for line ← 0, sizeof(mat) do
5:         for col ← (line + 1), sizeof(mat) do
6:                     ▷ Find the position of the highest value for each line
7:             if mat[line][col] > max then
8:                 max ← mat[line][col]
9:                 max_pos ← col
10:            end if
11:        end for
12:        if max_pos > 0 then
13:    ▷ Bring higher value column/line close to the matrix main diagonal
14:            for i ← 0, sizeof(mat) do
15:                aux = mat[i][line + 1]                    ▷ swap column
16:                mat[i][line + 1] = mat[i][max_pos]
17:                mat[i][max_pos] = aux
18:                aux = mat[line + 1][i]                    ▷ swap line
19:                mat[line + 1][i] = mat[max_pos][i]
20:                mat[max_pos][i] = aux
21:            end for
22:        end if
23:    end for
24: end function
```

proposal considers the visual representation of the TMs, which can be seen as a texture, as depicted in Fig. 2.

Texture classification techniques were explored by several authors in recent years, and as a result a great number of descriptors can be found in the literature, such as Grey Level Co-ocurrence Matrices (GLCM) [31], Gabor filters [32], Local Phase Quantization (LPQ) [33], and Local Binary Pattern (LBP) [21]. Among all these descriptors, the LBP gained significant popularity because of its high discriminative power and computational simplicity, which makes it possible to analyse images in challenging real-time settings. With that in mind, in this work, we used two optimizations of LBP as the texture descriptors.

For implementing LBP in DCTraCS, we consider *C* as each cell in the TM. The value of *C* is compared with its eight neighbors cells, starting with the top-left, and following a clockwise order. If *C* is greater than the neighbor's value, write "0", otherwise write "1", resulting in an 8-digit binary number which is converted to decimal, as exemplified in Fig. 3; this is the LBP value calculated for that cell. The same operation is performed with all cells in the TM generating a histogram as a 256-dimensional feature vector.

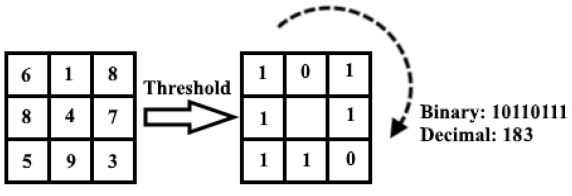


Fig. 3. The LBP operator

The first optimization of LBP implemented in this work is called ULBP [21], it introduces a concept based on the transition between 0's and 1's in the LBP image, reducing the length of the histogram to a 59-dimensional feature vector. A binary LBP code is considered uniform if the number of transitions is less than or equal to 2, also considering that the code is seen as a circular list. That is, the code 10110111 is not considered uniform because it contains four transitions. But the code 11110111 is characterized as uniform because it has only two transitions.

The second optimization is named RLBP, which considers the input image as being noisy and proposes changing one single bit from the original LBP, only if this modification turns it in a uniform pattern. In the previous example, if we change the second bit from 0 to 1 (1**1**110111), it will result in a uniform pattern, which is a more meaningful pattern for the texture representation and classification.

### D. Classifiers

We tested different supervised machine learning classifiers but the best results were produced by the SVM with Gaussian kernel [23] and the RF [24]. SVM is a popular classification algorithm, which builds a hyperplane in a high-dimensional space that may be used either for classification or regression. Different from other linear discriminant functions, it provides the optimal hyperplane for separating two classes.

The RF is an ensemble approach that uses decision tree predictors. The rationale behind ensemble methods is that a group of weak learners (in this case the decision trees) can come together to form a strong learner. One of the advantages of the RF is that they are quite fast and able to deal with unbalanced data.

## IV. Experimental Setup and Evaluation

For evaluating our proposal, we collected data from two different computing systems, an Intel based data center and a High-Performance Computing (HPC) machine. We extracted the TMs of 12 scientific applications and also the TMs of the two most expressive phases of MapReduce applications, comparing our approach with four methods used for application classification. The results and the discussion are reported throughout this section.

### A. Experimental Testbeds

*1) Intel Based (data center A):* This computing system is composed of 32 Lenovo PCs each with Intel quad-core 3.2Ghz processors, 8GB RAM, 1TB HD, 1 Gigabit Ethernet, running Linux Debian 8.2 and the latest available version of Disco MapReduce[2] (v0.5.4). These computers are connected to a 48 port Gigabit Ethernet Pica8 P-3290 switch running the operating system PicOS v2.6.4. This switch supports OpenFlow v1.4 through Open vSwitch[3] (v2.0) integration.

*2) BlueCrystal Phase 3 (data center B):* This system is a HPC machine belonging to the University of Bristol[4] which is comprised of 223 base blades, where each blade has 2.6 GHz SandyBridge processor with 16 cores, 64GB RAM, and a 1TB SATA disk. Besides these "base blades," there are also

[2]http://discoproject.org/
[3]http://openvswitch.org/
[4]https://www.acrc.bris.ac.uk/acrc/phase3.htm

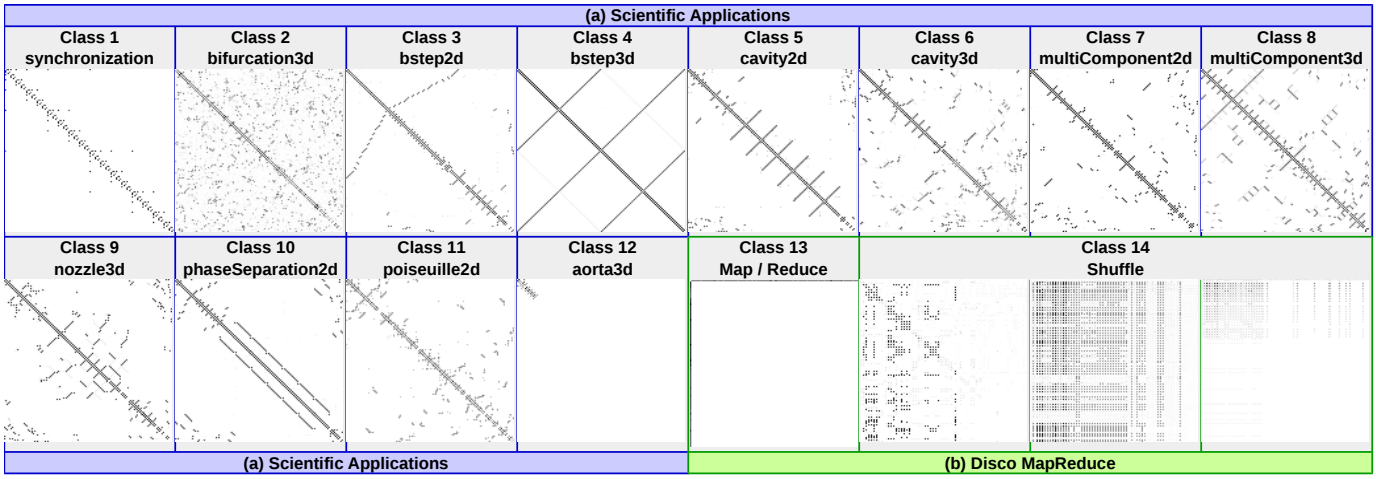


Fig. 4. Communication patterns of studied applications.

100 blades that can host dual GPGPUs, and 18 large memory blades each containing 256GB of memory. It runs Scientific Linux (v6.4), Torque (v4.2.4.1)[5] plus Moab (v7.2.9)[6] as the queuing system, and the mpich2 (v1.4.1p1)[7] as the standard MPI.

### B. Scientific Applications and MapReduce

As stated before, we tested our approach with two different types of applications commonly executed in data centers, ie, MapReduce and scientific applications. For creating the experimental dataset, we ran these applications on 128 nodes, collecting their TMs every second for a period of 30 minutes[8].

*1) Scientific Applications:* In our evaluation, we measured the TMs of scientific applications for simulating fluid flows by means of a lattice Boltzmann method, implemented using the OpenLB library [34]. We randomly selected 12 sample applications distributed with this open-source library; their TMs are shown in Fig. 4a, identified from *Class 1* to *Class 12*. All TMs presented in Fig. 4 were preprocessed, meaning that the unscrambling function was applied to them.

*2) MapReduce:* Is often used to solve problems when a vast amount of input information can be processed concurrently with a large number of computers (nodes). Usually, MapReduce takes advantage of data locality, processing it on or near the storage assets. In this computation phase, no communication occurred, and the collected TMs were "blank".

For evaluating MapReduce, we collected the TMs from Disco application examples[9]. After gathering the TMs, we analysed them and could identify two different classes of communications, which are presented in Fig. 4b. The operations Map and Reduce have a similar communication pattern (Class 13), presenting communication only from the master node to all other nodes and vice-versa. On the other hand, the data shuffling operation (Class 14) is network intensive where the communication randomly occurs among multiple pairs of nodes, not expressing a singular pattern, as shown in Fig. 4b.

### C. Gathering and Processing the TMs

For gathering the TMs in the *data center A*, we use the same method described in our previous work [28], installing the traffic measurement rules in each Software-Defined Networking (SDN)-enabled switch, while the SDN controller keeps periodically collecting the statistics for generating the traffic matrices; the gathering process took on average 0.34 seconds.

*Data center B* presented a challenge as BlueCrystal Phase 3 had certain restrictions in place, meaning that no permissions were given for accessing network devices. To combat this, we developed a new solution for collecting the TMs, using the *strace* Linux utility for intercepting all the network related system calls. These system calls were recorded to a file and parsed on each computing node for extracting the total amount of bytes transmitted to and received from all other nodes. At an adjustable time $t$, the computing nodes sent the parsed data to a master node which consolidated it into the TM for the time $t$. In this approach, the average time for acquiring the TMs was 0.51 seconds.

The average time for applying the unscrambling function was 35 µs and for extracting the feature vectors; the computing time was 34 µs for both ULBP and RLBP.

### D. Classification

For assessing the classification methods reported in this section, we used the following methodology. We executed each method ten times and, for every run, the input for the classifiers were created by randomly splitting the collected TMs in $2/3$ as the training set and $1/3$ as testing set, meaning that for each class, approximately 1200 entries were used for training and 600 entries for testing.

We compared our classification results with the state-of-the-art by extracting the feature vectors of four existing works that achieved significantly high classification accuracy:

[5]http://www.adaptivecomputing.com/products/open-source/torque/
[6]http://www.adaptivecomputing.com/products/hpc-products/moab-lite/
[7]https://www.mpich.org/
[8]This dataset is publicly available at: www.inf.ufpr.br/ctrois/dctracs/dataset/
[9]https://github.com/discoproject/disco/tree/develop/examples/

TABLE I
FEATURE VECTORS USED IN RELATED WORKS.

| Reference | Feature vector |
|---|---|
| Eerman et al. [17] | total number of packets<br>average packet length (client to server)<br>average packet length (server to client)<br>average packet length (bidirectional)<br>flow duration<br>average data packet length<br>average packet inter-arrival time |
| Fahad et al. [20] | server port<br>total number of packets with PUSH bit<br>total number of bytes sent in the initial window (server to client)<br>total number of RTT samples |
| Soysal and Schmidt [27] | server port number<br>client port number<br>total number of packets<br>total number of bytes<br>flow duration<br>service type<br>flags type<br>protocol type |
| Zhang et al. [19] | server port<br>minimum segment size (client to server)<br>total number of bytes sent in the initial window (server to client)<br>total number of bytes sent in the initial window (client to server) |

Eerman [17], Fahad [20], Soysal [27], and Zhang [19][10]; the information used for creating their respective feature vectors is presented in Table I.

For every experiment, we tuned the classifiers as follows. The kernel parameters $\gamma$ and $C$ for the Gaussian SVM were empirically defined through a grid search and fivefold cross-validation using the training set. This operation was also applied to find the number of trees for the RF classifier. All the experiments were carried out using scikit-learn [35], an open-source machine learning library in Python.

Fig. 5 shows the accuracy SVM and RF for classifying the applications. As can be seen, the best overall accuracy was obtained with visual textures extracted from the TMs. SVM achieved better results for classifying our approach, achieving 98.9% when the feature vector was created using RLBP and 99.0% with ULBP. The accuracy of other methods was higher with the RF classifier, with 75.8%, 87.6%, 61.6%, and 85.8% for Eerman, Fahad, Soysal, and Zhang, respectively.

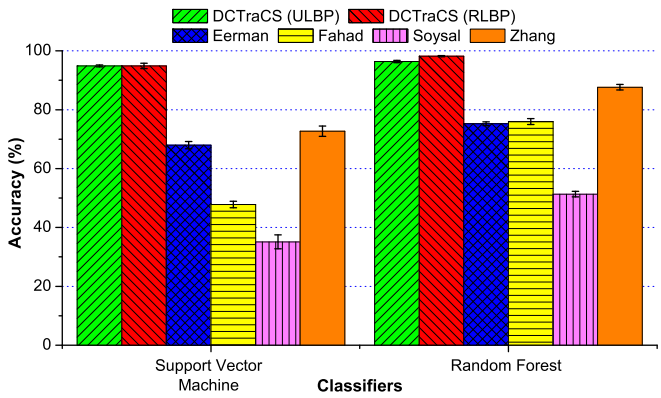


Fig. 5. Accuracy of classifiers.

We observed in Fig. 5, that the feature vectors described in the previously discussed literature (Eerman [17], Fahad [20], Zhang [19], and especially Soysal [27]) did not achieve the level of accuracy as reported in their respective papers. We believe that the contributing factor to this discrepancy was that all scientific applications use the same server port (TCP port number 22) for encrypted communication and, not being a discriminative feature for classifying these applications.

To evaluate output quality of our approach, we plotted the Receiver Operating Characteristic (ROC) curves of SVM classifier. This curve is created by plotting the *true positive rate* on the Y axis against the *false positive rate* on the X axis. The top left corner of the plot is the ideal point, with a false negative rate of zero, and a true positive rate of one, meaning that a larger area under the curve (AUC) is usually better. ROC curves are typically used in binary classification and, for plotting multiclass ROC curves, we computed the ROC curve for each class, comparing it against the other classes [36].

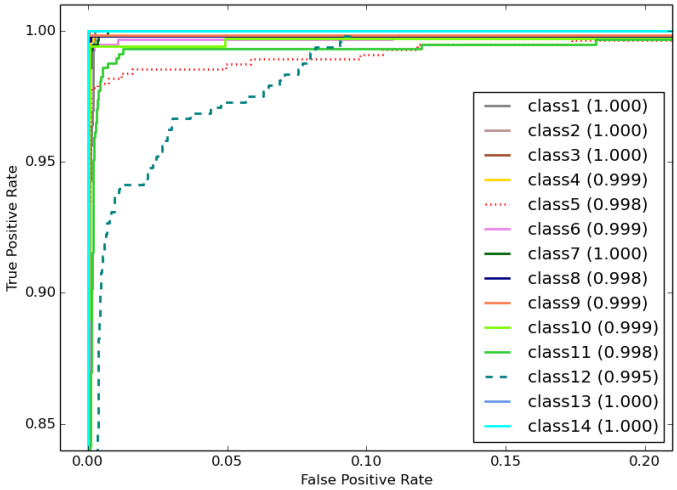


Fig. 6. ROC curve of DCTraCS using ULBP and SVM.

Fig. 6 shows the ROC curve of DCTraCS using ULBP. The values in parenthesis next to the classes' numbers show the AUC. The figure was enlarged to show the details of the upper left corner of the curves, showing that *Class 12* presented the most problems in the classification, followed by *Class 5*.

The confusion matrix (Fig. 7) shows that Class 12 was 27 times wrongly predicted as Class 1. This problem probably happened because, as can be seen in Fig. 4, Class 12 presents few textural information since this application exchanged information only across 16 of 128 available computational nodes. Class 5 was wrongly predicted 12 times, mostly being confused with Class 11. Despite these problems, the percentage of misclassification is relatively low.

Finally, Fig. 8 illustrates the average classification time normalised to the maximum value[11]. Our first observation is that RF was faster than SVM in all cases, this can be explained by the very different nature of the algorithms design. The results show that Eerman [17], Fahad [20], and Soysal [27] achieved

[10]For simplicity, when referring to these works, we used only the first author's surname.

[11]In our testbed, the highest classification time was 158 µs, computed when SVM classified DCTraCS (ULBP).

| Actual \ Predicted | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 | C13 | C14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C1 | 514 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C2 | 0 | 484 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C3 | 0 | 0 | 198 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C4 | 0 | 0 | 0 | 478 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| C5 | 0 | 0 | 1 | 0 | 539 | 1 | 1 | 0 | 0 | 1 | 8 | 0 | 0 | 0 |
| C6 | 0 | 0 | 0 | 0 | 0 | 536 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C7 | 0 | 0 | 0 | 0 | 2 | 1 | 560 | 0 | 0 | 0 | 5 | 0 | 0 | 0 |
| C8 | 0 | 0 | 1 | 0 | 0 | 3 | 0 | 500 | 0 | 0 | 0 | 0 | 0 | 0 |
| C9 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 543 | 0 | 0 | 0 | 0 | 0 |
| C10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 344 | 0 | 0 | 0 | 0 |
| C11 | 0 | 0 | 0 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 560 | 1 | 0 | 0 |
| C12 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 446 | 0 | 0 |
| C13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 115 | 0 |
| C14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 148 |

Fig. 7. Confusion matrix of DCTraCS using ULBP and SVM.

the highest classification speeds with the RF classifier, slightly higher than 20% of the highest measured time.

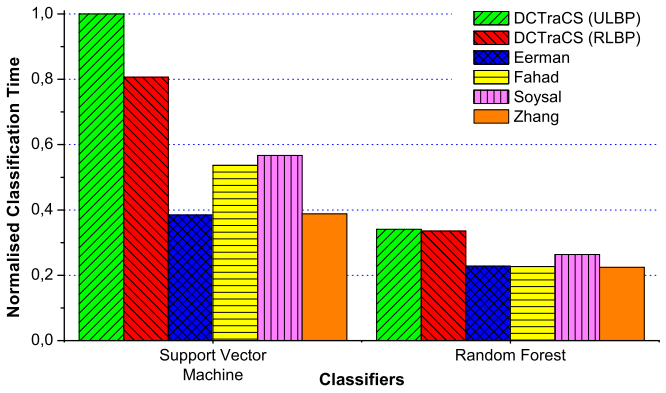


Fig. 8. Normalized classification time.

It is possible to observe a difference of 20% when SVM classified the two implementations of LBP. This variation occurred because, during the training phase, the SVM needed more support vectors for ULBP than for RLBP. With SVM the fastest time was for Eerman [17], followed by Zhang, taking just under 40% of the maximum time. When comparing the different approaches with the RF classifier, we can see that DCTraCS is about 10% slower than the other methods.

### *E. Discussion*

The time measured for RF to classify DCTraCS is around 10% higher than that of other approaches. This difference in our testbed corresponds to 17.8 µs; considering that a data center application may take many hours for executing, this time is negligible. On the other hand, the accuracy of our method is less than 1% of the optimal solution, and the best accuracy obtained with the other methods was 11.3 percentage points worse than ours.

Another important consideration is the time for executing the entire process, from acquiring the TMs to classifying the applications. In the worst case, the total time is around 0.7 seconds. So, if we consider that the execution time of a MapReduce or a scientific application may take hours, we perceive that it is feasible using DCTraCS for traffic classification. It can be attached, for instance, to a SDN controller for modifying the network forwarding to improve the applications performance or enabling a network orchestrator in a data center environment for properly placing the working nodes.

Finally, the last point to be considered is that texture is a powerful tool for classification. It has been used for classifying hundreds of classes in many different fields. For instance, Bertolini et al. [37] has used the textures for identifying and verifying 650 different writers, achieving accuracies as higher as 99%.

## V. Conclusion

In this paper, we proposed a novel approach for classifying data center traffic using the textures of TMs as input for ML techniques. Our method consisted of measuring the TMs and applying an unscrambling function for generating different image textures for each application. The feature vectors were extracted from the images through the ULBP and the RLBP, two methods widely used for texture recognition.

We implemented the proposed method in a system called DCTraCS, and we employed two classifiers, SVM and RF, for evaluating it. We performed experiments for classifying the traffic for four scientific applications plus two MapReduce execution phases, comparing the accuracy of our proposal with four methods described in current literature. As scientific applications use secure communication and the Shuffle phase of MapReduce presents a high level of randomness in the communications between the computational nodes, the accuracy of the other approaches were lower than our approach. On the other hand, our proposal uses information extracted from the TMs and therefore has the advantage of having a global and comprehensive view of the network communications.

We concluded that it is possible to use different information from that commonly used (i.e. intrinsic and statistical flow information) for traffic classification and, as the results presented in this paper showed, the existing texture from TMs images may be used as a robust representation for building a classification system. We plan to extend our current work by firstly utilizing DCTraCS within a live data center or similar environment during its normal day to day operation to classify live traffic in real time. Secondly, we want to create a self-adapting mechanism, using the input data as feedback for the training set, allowing the classification process to evolve over time.

## Acknowledgments

The authors would like to thank CAPES for partial funding of this research, CNPq under Grant no. 456143/2014-9, the Brazilian Ministry of Communications for partial funding it via “Digital Inclusion: Technology for Digital Cities” project, the Brazilian Ministry of Science, Technology, Innovation, and Communication (MCTIC) through RNP, CTIC and FAPES (under grants no. 0410/2015 and no. 0444/2015), and the

UFSM/FATEC through project number 041250 - 9.07.0025 (100548). We also thank the European Commission H2020 program for partially funding this work under grant agreement no. 688941 (FUTEBOL).

## REFERENCES


[1] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 92–99, Jan. 2010.

[2] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams *et al.*, "The landscape of parallel computing research: A view from berkeley," Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Tech. Rep., 2006.

[3] S. Werner, J. Navaridas, and M. Luján, "Designing low-power, low-latency networks-on-chip by optimally combining electrical and optical links," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 265–276.

[4] J. Wang, N. Rubin, A. Sidelnik, and S. Yalamanchili, "Laperm: Locality aware scheduler for dynamic parallelism on gpus," in *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 2016, pp. 583–595.

[5] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, and K. Olukotun, "Plasticine: A reconfigurable architecture for parallel paterns," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017, pp. 389–402.

[6] E. Rubin, E. Levy, A. Barak, and T. Ben-Nun, "Maps: Optimizing massively parallel applications using device-level memory abstraction," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 4, pp. 1–22, Dec. 2014.

[7] G. Srivastava, M. Singh, P. Kumar, and J. Singh, "Internet traffic classification: A survey," in *Recent Advances in Mathematics, Statistics and Computer Science*. World Scientific, 2016, pp. 611–620.

[8] V. B. Iversen, "Teletraffic engineering and network planning," 2015.

[9] C. Trois, M. Martinello, L. C. E. de Bona, and M. D. Del Fabro, "From software defined network to network defined for software," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ser. SAC '15. ACM, 2015, pp. 665–668.

[10] D. Jiang, Z. Xu, P. Zhang, and T. Zhu, "A transform domain-based anomaly detection approach to network-wide traffic," *Journal of Network and Computer Applications*, vol. 40, pp. 292–306, 2014.

[11] N. Promrit and A. Mingkhwan, "Traffic flow classification and visualization for network forensic analysis," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, March 2015, pp. 358–364.

[12] M.-S. Kim, Y. J. Won, and J. W.-K. Hong, "Application-level traffic monitoring and an analysis on ip networks," *ETRI journal*, vol. 27, no. 1, pp. 22–42, 2005.

[13] L. A. Rocha and F. L. Verdi, "A network-aware optimization for vm placement," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, March 2015, pp. 619–625.

[14] M. Caria, A. Engelmann, A. Jukan, and B. Konrad, "How to slice the day: Optimal time quantization for energy saving in the internet backbone networks," in *2012 IEEE Global Communications Conference (GLOBECOM)*, Dec 2012, pp. 3122–3127.

[15] Y. Dhote, S. Agrawal, and A. J. Deen, "A survey on feature selection techniques for internet traffic classification," in *Computational Intelligence and Communication Networks (CICN), 2015 International Conference on*. IEEE, 2015, pp. 1375–1380.

[16] T. Bujlow, V. Carela-Español, and P. Barlet-Ros, "Comparison of deep packet inspection (dpi) tools for traffic classification," Universitat Politècnica de Catalunya, Tech. Rep., 2013.

[17] J. Eerman, A. Mahanti, and M. Arlitt, "Internet traffic identification using machine learning techniques," in *Proc. of the 49th IEEE Global Telecomm. Conf.(GLOBECOM)*, 2006, pp. 1–6.

[18] Y. Wang, Y. Xiang, and S. Z. Yu, "Automatic application signature construction from unknown traffic," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, April 2010, pp. 1115–1120.

[19] H. Zhang, G. Lu, M. T. Qassrawi, Y. Zhang, and X. Yu, "Feature selection for optimizing traffic classification," *Computer Communications*, vol. 35, no. 12, pp. 1457–1471, 2012.

[20] A. Fahad, Z. Tari, I. Khalil, A. Almalawi, and A. Y. Zomaya, "An optimal and stable feature selection approach for traffic classification based on multi-criterion fusion," *Future Generation Computer Systems*, vol. 36, pp. 156–169, 2014.

[21] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 7, pp. 971–987, 2002.

[22] Y. Zhao, W. Jia, R.-X. Hu, and H. Min, "Completed robust local binary pattern for texture classification," *Neurocomputing*, vol. 106, pp. 68 – 76, 2013.

[23] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer Science & Business Media, 1999.

[24] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[25] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Comm. of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

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

[27] M. Soysal and E. G. Schmidt, "Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison," *Performance Evaluation*, vol. 67, no. 6, pp. 451–467, 2010.

[28] C. Trois, L. C. E. de Bona, M. D. D. Fabro, M. Martinello, S. Bidkar, R. Nejabati, and D. Simeonidou, "Softening up the network for scientific applications," in *Parallel, Distributed and Network-Based Processing (PDP), 25th Euromicro Inte. Conference on*. IEEE, 2017, pp. 410–418.

[29] Y. Gong, X. Wang, M. Malboubi, S. Wang, S. Xu, and C.-N. Chuah, "Towards accurate online traffic matrix estimation in software-defined networks," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. ACM, 2015, p. 26.

[30] C. Trois, L. C. E. de Bona, M. D. D. D. Fabro, and M. Martinello, "Carving Software-Defined Networks for Scientific Applications with SpateN," in *41st Conference on Local Computer Networks (LCN)*. IEEE, 2016, pp. 606–610.

[31] R. M. Haralick, "Statistical and structural approaches to texture," *Proceedings of IEEE*, vol. 67, no. 5, 1979.

[32] M. Lyons, S. Akamatsu, M. Kamachi, and J. Gyoba, "Coding facial expressions with gabor wavelets," in *3rd IEEE International Conference on Automatic Face and Gesture Recognition*, 1998, pp. 200–205.

[33] V. Ojansivu and J. Heikkila, "Blur insensitive texture classification using local phase quantization." in *Proc. Image and Signal Processing (ICISP 2008)*, 2008, pp. 236–243.

[34] V. Heuveline and J. Latt, "The openlb project: an open source and object oriented implementation of lattice boltzmann methods," *International Journal of Modern Physics C*, vol. 18, no. 04, pp. 627–634, 2007.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[36] D. J. Hand and R. J. Till, "A simple generalisation of the area under the roc curve for multiple class classification problems," *Mach. Learn.*, vol. 45, no. 2, pp. 171–186, Oct. 2001. [Online]. Available: https://doi.org/10.1023/A:1010920819831

[37] D. Bertolini, L. S. Oliveira, E. Justino, and R. Sabourin, "Texture-based descriptors for writer identification and verification," *Expert Systems with Applications*, vol. 40, no. 6, pp. 2069–2080, 2013.