# Realtime Classification for Encrypted Traffic

Roni Bar-Yanai[1], Michael Langberg[2,⋆], David Peleg[3,⋆⋆], and Liam Roditty[4]

[1] Cisco, Netanya, Israel
rbaryana@cisco.com
[2] Computer Science Division, Open University of Israel, Raanana, Israel
mikel@openu.ac.il
[3] Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel
david.peleg@weizmann.ac.il
[4] Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel
liamr@macs.biu.ac.il

**Abstract.** Classifying network flows by their application type is the backbone of many crucial network monitoring and controlling tasks, including billing, quality of service, security and trend analyzers. The classical "port-based" and "payload-based" approaches to traffic classification have several shortcomings. These limitations have motivated the study of classification techniques that build on the foundations of learning theory and statistics. The current paper presents a new statistical classifier that allows real time classification of encrypted data. Our method is based on a hybrid combination of the $k$-means and $k$-nearest neighbor (or $k$-NN) geometrical classifiers. The proposed classifier is both fast and accurate, as implied by our feasibility tests, which included implementing and intergrading statistical classification into a realtime embedded environment. The experimental results indicate that our classifier is extremely robust to encryption.

## 1 Introduction

Classifying network flows by their application type is the backbone of many crucial network monitoring and controlling tasks. Basic network management functions such as billing, quality of service, network equipment optimization, security and trend analyzers, are all based on the ability to accurately classify network traffic into the right corresponding application.

Historically, one of the most common forms of traffic classification has been the *port-based* classification, which makes use of the port numbers employed by the application on the transport layer. However, many modern applications use dynamic ports negotiation making *port-based* classification ineffective [10,17] with accuracy ranges between 30% and 70%.

The next step in the evolution of classification techniques was *Deep Packet Inspection* (DPI) or *payload-based* classification. DPI requires the inspection of

⋆ Supported in part by The Open University of Israel's Research Fund (grant no. 46109) and Cisco Collaborative Research Initiative (CCRI).
⋆⋆ Supported in part by Cisco Collaborative Research Initiative (CCRI).

P. Festa (Ed.): SEA 2010, LNCS 6049, pp. 373–385, 2010.

the packets' payload. The classifier extracts the application payload from the TCP/UDP packet and searches for a signature that can identify the flow type. Signatures usually include a sequence of bytes/strings and offsets that are unique to the application and characterize it. DPI is widely used by today's traffic classifier vendors. It is very accurate [11,17] but suffers from a number of drawbacks.

Recently, we have witnessed a dramatic growth in the variety of network applications. Some of these applications are transmitted in an encrypted manner, posing a great challenge to the DPI paradigm. Such applications may choose to use encryption both for security and to avoid detection. Common P2P applications such as BitTorrent and eMule have recently added encryption capabilities (primarily to avoid detection). As a significant share of the total bandwidth is occupied by P2P applications and since current DPI based classifiers must see the packet's payload, encryption may become a real threat for ISP's in the near future. The inability of port-based and payload-based analysis to deal with the wide range of new applications and techniques used in order to avoid detection has motivated the study of other classification techniques. Two examples include *behavior* based classification and classification based on a combination of learning theory and statistics.

In the *behavioral* paradigm, traffic is classified by identifying a certain behavior that is unique to the application at hand. In this setting, the signature is not syntactic (as in DPI classification) but rather a combination of events. For example, it is possible to identify encrypted BitTorrent by intercepting the torrent file (a file used to start the downloading process in BitTorrent clients) [3]. The torrent file includes a list of BitTorrent hosts, each possessing certain parts of the downloaded file. The classifier processes the file and saves these hosts. Now, an encrypted flow that is destined to one of these hosts will be marked immediately as BitTorrent. The drawback of such behavioral solutions is that they are too specific, and it will not take long before the P2P community strikes back by encrypting torrent files [3].

This paper presents a new method for statistical classification. Our method is based on a hybrid combination of the well known $k$-means and $k$-nearest neighbor geometrical classifiers. The proposed classifier is both fast and accurate, as implied by our feasibility tests, which included implementing and intergrading our classifier into a realtime embedded environment. The experimental results indicate that our classifier is extremely robust to encryption and other DPI flaws, such as asymmetric routing and packet ordering. Finally, we show how to boost the performance of our classifier even further, by enhancing it with a simple *cache-based* mechanism that combines elements of port-based and statistical classification. In what follows, we elaborate on statistical classification in general and specify our contribution.

*Related work.* The statistical approach to classification is based on collecting statistical data on properties of the network flow, such as the mean packet size, flow duration, number of bytes, etc. The statistical paradigm relies on the assumption that each application has a unique distribution of properties that represents it

and can be used to identify it. This approach has been the subject of intensive research in the recent years.

The work of Paxson [15] from 1994 established a relationship between flow application type and flow properties (such as the number of bytes and flow duration). A methodology for separating chat traffic from other Internet traffic, which uses statistical properties such as packet sizes, number of bytes, duration and inter arrival times of packets, was developed in [5]. Mcgregor et al. [12] explored the possibility of forming clusters of flows based on flow properties such as packet size statistics (e.g., minimum and maximum), byte count, and idle times etc. Their study used an *expectation maximization (EM)* algorithm to find the clusters' distribution density functions.

A study focusing on identifying flow application *categories* rather than specific individual applications was presented in [16]. While limited by a small dataset, they showed that the $k$-nearest neighbor algorithm and other techniques can achieve rather good results, correctly identifying around 95% of the flows. Zander *et al.* [18], using an EM based clustering algorithm, obtained an average success rate of 87% in the separation of individual applications. The basic Navie Bayes algorithm, enhanced by certain refinements, was studied by Moore *et al.* [13] and was shown to achieve an accuracy level of 95%.

*Realtime* classification, in which the flow is to be classified based mainly on its first few packets' size and direction, was addressed in [2,4,7]. It is important to note that these algorithms were tested only against basic application protocols. Encrypted BitTorrent and Gnuttela, for example, use packet padding in the beginning of the flow start, to avoid such detection methods. For more details on related work see [10,14].

*Our contribution.* The current paper introduces a hybrid statistical algorithm that integrates two basic and well known machine learning algorithms, known as $k$-nearest neighbors and $k$-means. The algorithm is fast, accurate and most important it is insensitive to encrypted traffic. Moreover, the strength of our algorithm is precisely in overcoming several weaknesses of the DPI approach, which is the leading technology used by current network classifiers. In particular, our algorithm overcomes asymmetric routing[1] and packet ordering. To the best of our knowledge, our study is the first to demonstrate the potential of statistical methods on encrypted traffic in realtime classification.

To put our results in perspective, we note that most previous statistical classification methods were tested in an off-line environment [14]. The results on realtime classification [2,4,7] are all based on inspecting the first five initial packets of the flow, and thus work well only when these packets represent the application under study. Note that encrypted Bittorrent and EMule, which use padding on their initial packets, cannot be classified using such techniques.

The strength of our algorithm is demonstrated on Encrypted BitTorrent, one of the hardest applications to identify. The BitTorrent development community puts a lot of effort into detection avoidance and uses port alternation, packet padding (on initial flow packets) and encryption as part of this effort. Actually, as

[1] Occurring when incoming and outgoing flows use different routers.

our algorithm is insensitive to encryption, it turns out that it identifies encrypted and non-encrypted BitTorrent flows with the exact same accuracy.

The data set used for the experiments reported in this paper was recorded in 2009 (full payload) on a real ISP network edge router on two different geographical locations, and contains millions of flows. The record is unique in its relevance and reflects the distribution and behavior of contemporary application flows. We integrated our statistical classifier into a realtime embedded environment. The feasibility test included a full implementation of our algorithm on SCE2020, which is one of the leading Cisco platforms specialized to classification. The algorithm was tested in full line rate, and the experiment has demonstrated that our algorithm can be implemented and integrated on platforms that are limited on resources, memory and cpu, and require realtime responses.

In addition, we show that using a simple LRU cache drastically reduces classification time and memory of more than 50% of the flows, and also increases the classification accuracy of some of the protocols.

## 2 Methodology

The general paradigm followed by our classifier is a machine learning one. Roughly speaking, we first build a training set and use it to train our classifier; we then turn to the task of classification. In what follows we briefly elaborate on the techniques we use to obtain labeled traffic (for our training set), we then address some special properties of the data sets we use.

*Collecting labeled data.* To train our classifier, we require a collection of reliably classified traffic flows[2]. We were provided such a database generated using Endace [6] for real-time traffic recording and injecting, and labeled using the Cisco SCE 2020 box (a professional tool for classifying and controlling network traffic) coupled with manual inspection and verification. The database included 12 million flows recorded in 2009 on ISP network edge routers on two different geographical locations.

As mentioned earlier, one of the major challenges in flow classification is identifying encrypted flows. In the current study we used two different sources to obtain encrypted flows. Our first database, which was recorded in 2009, contains some encrypted flows of BitTorrent and Skype. Our second source was a manual recording of a BitTorrent application taken in a controlled environment.

*Special properties of the data set.* Short flows, namely, flows with fewer than 15 payload packets, were removed from the dataset. The rational behind ignoring short flows is that we are using the statistical properties of the flow for classification, and measuring such properties on short flows is unreliable. Hence short flows require a different approach. Note that in many practical scenarios, classifying short flows is of lower priority, as they account for an insignificant

[2] The term *flow* refers to a single data flow connection between two hosts, defined uniquely by its five-tuple (source IP address, source port, destination IP address, destination port, protocol type TCP/UDP).

fraction of the overall utilized bandwidth. We remark that flows with fewer than 15 packets account for 87% of the total flows but only 7% of the total bytes. Using the algorithm refinement of an LRU cache for heavy hosts, we were able to classify around fifty percent of the short flows as well, thus reducing the total bytes that were actually ignored to approximately 3.5%.

Another property of the data set is that only applications with sufficiently significant representation in the data traces were considered. Specifically, we considered applications that had at least 4000 flow instances in our records. The flow distribution was as follows: Http flows accounted for 59% of the flows, BitTorrent for 17.1%, SMTP 13%, EDonkey for 8.5%, and POP3, Skype, Encrypted BitTorrent, RTP and ICQ were each responsible for less than 1% of the flows.

## 3 The Classification Algorithm

We now specify our machine learning based classification algorithm. The host initiating the flow is defined as the *client* and the host accepting the flow - as the *server*. We consider only packets that contain payload.

*Feature extraction.* The use of classification algorithms based on machine learning requires us to parameterize the flow, turning each flow $x$ into a vector of features $\bar{V}_x = \langle V_1, \ldots, V_d \rangle$, where each coordinate $V_i$ contains some statistical parameter of the flow $x$ (e.g., its packet mean size). Our study focused on real-time classification, making it necessary to concentrate on features that are both cheap to calculate and can be calculated in streaming mode (namely, inspecting a single packet at a time and seeing each packet only once).

The feature extraction stage consists of two phases. In the first phase, we consider basic traffic flow properties and collect the corresponding parameters for each flow. The statistics are collected until we reach *classification point* (the point in time upon we decide on the flow's application type). All the experimental results reported in this paper used an inspection length parameter of $m = 100$ packets, that is, all flows were classified upon seeing packet 100 (or earlier, if the flow size was less than 100 packets). In the second phase, once the classifier reaches classification point, it turns the statistics collected into a feature vector, which is then used as the input for the classifier.

*Feature Set.* Our complete feature set included the following 17 different parameters: Client number of packets; Server number of packets; Total number of packets; Client packet size expectation; Server packet size expectation; Client average 'packets per second' rate; Server average 'packet per second' rate; Client packet size variance; Server packet size variance; Total client bytes; Total server bytes; Download to upload ratio; Server average number of bytes per bulk[3]; Client average number of bytes for bulk; Server average number of packets for bulk; Client average number of packets for bulk; and Transport protocol (TCP or UDP). Note that in the asymmetric setting, some of our features take zero value. Moreover, unidirectional flows exist in a symmetric routing as well, for example during FTP download.

[3] Contiguous parts of a flow, separated by idle periods of 1sec or more.

*The k-nearest neighbors algorithm.* This is one of the simplest and most well-known classification algorithms. It relies on the assumption that nearby data sets have the same label with high probability. In its simplest form, the algorithm classifies flows as follows. Upon receiving the feature vector $\bar{V}_x$ of a new flow $x$, the algorithm finds its $k$-nearest neighboring flows (in Euclidean distance) in the training set. The flow $x$ is then assigned the label associated with the majority of those neighbors. In our experiments we used single neighbors ($k = 1$); we discuss the use of multiple neighbors in the discussion section. Our preliminary tests show a reasonably good classification rate of above 99% for $k = 1$.

The main problem with the $k$-nearest neighbors algorithm is that its time complexity grows linearly with the training set size, which is problematic as the training set may contain thousands of samples. Algorithm accuracy, learning time and complexity are compared in [10]. To address this disadvantage, we combined the $k$-nearest neighbors algorithm with the $k$-means algorithm.

*The k-means algorithm.* Another component in our classifier is the $k$-means algorithm. In the training phase of $k$-means classification, the flows are divided into $k$ clusters (according to geometrical similarity of their corresponding vectors). We then label each cluster based on the majority of flow types that have been assigned to the cluster. Now, a new flow $x$ is classified by finding the cluster $C$ whose center is nearest to $x$. The flow $x$ is assigned with the label of $C$. The algorithm's accuracy is only 83%, but it requires considerably less computational resources compared to the $k$-nearest neighbors algorithm.

An analysis of the distances between flows and their cluster centers[4] reveals that the distance distribution is Gaussian, so in each cluster most of the flows are placed close the cluster center. This suggests that the flow's nearest neighbor is likely to fall in the same cluster with high probability, and only instances very far from the cluster center can have their nearest neighbor placed in another cluster. Our hybrid algorithm, presented next, relies on this fact. We start by presenting the hybrid algorithm as a whole, and then discuss its properties.

*Our hybrid algorithm.* The core of our final classifier is a hybrid algorithm that integrates the above two algorithms, thus combining the light-weight complexity of the $k$-means algorithm, with the accuracy of the $k$-nearest neighbors algorithm. The hybrid algorithm also features additional refinements in the $k$-means clustering phase. As mentioned previously, our algorithm has two phases: the training phase and the classification phase.

In the training phase, using the labeled data in our training set, we construct a set of clusters in two stages. In the first stage, for each protocol (HTTP, SKYPE, ...), we run the $k$-means algorithm on the flows in our training set that are labeled as the protocol being considered. We note, that it is common that a collection of flows all generated by the same protocol may have very diverse behavior (and thus a diverse cluster structure). This follows by the fact that certain protocols (such as HTTP) may behave in different manners depending on the precise setting in which they are used (e.g., a HTTP flow carrying streaming

[4] Omitted for space considerations; also noted independently in [2], although different parameters were used.

video might not look similar to one carrying text only). The result of our first stage clustering, is a set of cluster centers ($k$ centers for each protocol), where each cluster is labeled naturally by the protocol in which it was constructed.

We now turn to the second stage of our clustering. After stage 1, clusters generated from different protocols might overlap, which might cause classification errors. To overcome this, in our second clustering stage, we redistributes the entire sample set of cluster centers defined in stage 1. Namely, using the same cluster centers that were found in stage 1, each flow in our training set is associated with the cluster center closest to it. Our two stage clustering is presented as Algorithm 1 below. In what follows, $X_i$ is the dataset of flows generated by application $i$, $C_i$ is the set of $k$ cluster centers of application $i$, and $C$ is the set of centers after stage 1 of our clustering ($k$ centers for each application).

**Algorithm 1.** Pseudo code for our two stage clustering

**for-each** $X_i \in \{X_1, ..., X_l\}$
    $C_i =$ **k-means**$(X_i, k)$
$C = C_1 \bigcup C_2 \bigcup .... \bigcup C_l$
$X = X_1 \bigcup X_2 \bigcup .... \bigcup X_l$
**for-each** $x_i \in X$
    associate $x_i$ with the closest center from $C$.

This concludes the training phase of our algorithm. Now, for a given flow $x$, the online classification is also done in two stages. First, we find the cluster center $c$ nearest to (the geometrical representation of) $x$. Note that this may not be enough. Namely, recall that after the second stage of our training, the clusters may not be homogeneous, and thus it is not clear how to label $x$ given $c$. For this reason, we use the second stage of our online classification, which runs the $k$-nearest neighbors algorithm (for $k = 1$) over the members of the cluster corresponding to $c$. The resulting Algorithm 2 is presented below.

**Algorithm 2.** Pseudo code for hybrid classification

$j = \text{argmin}_j \|x - c_j\|$, where $c_j \in C$
$nb = \text{argmin}_{x_i} \|x_i - x\|$, where $x_i$ is associated with cluster center $c_j$
**return** label($nb$)

Some remarks are in order. The design of our algorithm was guided by the observation that nearest neighbor classification is very accurate but slow in running time, while $k$-means classification is fast but has relatively weak accuracy. This naturally leads to the idea of combining the two algorithms. However, one may first consider a seemingly more natural way of combining the two algorithms, namely, for training take the entire training set and cluster it using the $k$ means algorithm (here, one would take a large $k$), and then perform the two-stage classification suggested above. We have checked this simpler hybrid technique, and indeed it yields very good results. Namely, on the one hand the accuracy

remains almost identical to that of the $k$-nearest neighbor algorithm, while on the other, the performance resembles that of the $k$-means algorithm. However, we have noticed that our two-stage training technique improves the overall accuracy (without modifying the running time). This follows from the fact that in our two-stage clustering, flows of the same protocol tend to be clustered together. Thus in classification, using the nearest neighbor approach, we are able to ovoid mislabelings. We also note that our two-stage training procedure is more efficient in running time than the naive single stage training, despite the fact that we run the $k$-means algorithm multiple times (once for each application type). This can be explained by the use of a much smaller data set on each separate run.

*Setting parameters and running time analysis.* The complexity of the algorithm highly depends on the clusters that were formed and on the distribution of the inspected flows. In the worst case, the outcome could be an unbalanced clustering, with few large clusters and many small ones. The hybrid algorithm uses the nearest-neighbor procedure within the nearest cluster as the final stage of classification, and therefore the complexity is directly affected by the cluster size. It is not unreasonable to assume that the distribution of the inspected flows correlates strongly with the cluster distribution, i.e., most of the flows will likely be assigned to a large cluster. In this case we lose accuracy without achieving the desired performance improvement.

We overcome this difficulty by setting a maximum size for each cluster. Then, in the end of our training phase, we reduce the size of a large cluster by removing random flows from it until reaching the desired size. We found it useful to bound cluster sizes by $4n/c$, where $c$ is the number of clusters and $n$ is the training set size. Our experiments show that this restriction does not affect the overall accuracy. On the other hand, the complexity of our classification can now be bounded by $4\frac{n}{c} + c$. Namely, it takes $c$ comparisons to find the closest cluster center, and then $4\frac{n}{c}$ comparisons to find the nearest neighbor in the cluster at hand. We minimize the running time of $4\frac{n}{c} + c$ by setting $c$ to $4\sqrt{n}$.

The complexity of the hybrid algorithm is thus much better than that of the nearest neighbors algorithm (which is $n$). In fact, this is just a worst-case upper bound, and the actual experimental results are even better; as seen in Section 4, the practical complexity is almost as good as the complexity of $k$-means.

*Leveraging Internet "heavy host" nature to save performance.* Another useful component of our hybrid classifier makes use of a simple and relatively small cache in order to save more than 50% of the classification time and memory. This component relies on the *heavy host* phenomenon. Heavy hosts are hosts that consume considerably more network resources compared to other hosts. Both the Web and P2P systems are known to have heavy hosts [1,8,9]. Inspecting P2P and HTTP traffic usually reveals a small percentage of hosts that account for a large percentage of the total flows and used bandwidth. In the Web, this behavior is mainly driven by content popularity [8], as popular content is often held at a small number of servers. In P2P systems, heavy hosts behavior is caused by a different reason, namely, the distribution of P2P traffic, which is dominated by

"free riders" (i.e., clients that do not contribute content and mainly consume) and "benefactors" (i.e., clients that mainly contribute and do not consume) [1].

Our findings show that the top ranked host accounts for about 0.7% of the total flows, and the same more or less goes for the second ranked host. The third accounts for 0.5%, the tenth accounts for 0.3% and these figures continue to drop sharply. All in all, we find that the 1000 top ranked hosts account for more than 50% of the flows. This distribution suggests the use of an LRU cache in our classification process. For each server stored in the LRU cache, the cache keeps the host IP and port of the flow server as its key and the flow class as the value. The classification works as follows. On receiving a new flow, first check if its server's host IP and port are stored in the LRU. If the information exists in the cache, then classify the flow according to the LRU cache, else use the classification algorithm and store the result in the LRU cache.

This classification caching scheme has a serious flaw: if the algorithm misclassifies one of our top ranked servers, then all flows destined to it would be misclassified as well. To overcome this problem, we keep in the LRU cache the last $\ell$ classification results destined to a given server, for some parameter $\ell$. Once we have $\ell$ results concerning a given server in the LRU cache, we apply a majority vote to decide its class. This improves our accuracy, as the probability of misclassification drops exponentially with $\ell$. Misclassification can be reduced even further by adding checkpoints, and rerunning the classification algorithm every $\rho$ classifications, replacing the oldest classification in the last classification list. This improves the overall accuracy of the LRU cache by an additional 1%-2%. Our experiments indicate that using the LRU cache, more than half of the flows are classified on the basis of their first packet, in $O(1)$ time. Finally, we remark that it is possible to use such LRU caching within any classifier to boost both its performance and accuracy.

## 4 Results

In this section we present our evaluation methods and the experimental results obtained by our algorithms. We also discuss a unique aspect of our work, namely, the implementation and testing of our algorithm in line rate in the SCE 2020, the network traffic controller box of Cisco.

*Algorithm evaluation.* We used two data sets in our validation process, one small and the other much larger. For the small data set we extracted 4000 flows from each application type (32k flows in total). On each test we partitioned the data set into two: a training set consisting of 1000 randomly selected flows and a validation set (to be classified) containing the rest of the flows. We repeated the test several times and took the average result. For the large data set we used the entire data set available, where again 1000 flows of each application type were chosen randomly into the training set and the rest of the flows were taken into the validation set (1.5M flows in total). The basic experiments were done using the small data set, while some of the major experiments were repeated on the large data set. The results were very consistent, and the main added value

of the large data set turned out to be the ability to test one of our algorithm refinements, namely, the use of the LRU cache.

*Algorithm accuracy.* Our results are presented in Table 1. BitTorrent (BT) and Encrypted BitTorrent flows were grouped together, namely, classifying encrypted BitTorrent into non-encrypted BitTorrent was counted as a success and the same for the other way around. As mentioned, the $k$-means algorithm is somewhat less accurate and achieves a modest average accuracy rate of 83%. The $k$-nearest neighbors ($k$-NN) algorithm achieves the best results, with overall accuracy of 99.1%. Its accuracy on traditional applications is very close to 100%, but as mentioned, the algorithm is too expensive for realtime applications. The accuracy of the hybrid algorithm is very similar to that of the $k$-nearest neighbors algorithm, implying that very little accuracy is lost by combining the two approaches.

**Table 1.** Algorithm accuracy

| Algorithm | Http | SMTP | POP3 | Skype | EDonkey | BT | Encrypted BT | RTP | ICQ |
|---|---|---|---|---|---|---|---|---|---|
| $k$-means | 0.78 | 0.93 | 0.93 | 0.85 | 0.80 | 0.75 | 0.74 | 0.93 | 0.71 |
| $k$-NN | 0.997 | 0.999 | 1.0 | 0.945 | 0.947 | 0.96 | 0.98 | 0.997 | 0.962 |
| hybrid | 0.997 | 0.999 | 0.998 | 0.94 | 0.94 | 0.963 | 0.974 | 0.992 | 0.954 |

One of the main purposes of this study was dealing with encrypted flows in realtime. Indeed, encrypted BitTorrent exhibited results similar to non-encrypted BitTorrent. Also note that Skype (which is encrypted) exhibits an accuracy similar to BitTorrent. These results look very promising and indicate that our algorithm is insensitive to encryption and can classify encrypted traffic as easily as non-encrypted one. The experiments conducted using our own generated records (recorded manually, see Section 2) yielded even a higher accuracy. We note that this may be attributed in part to localization effects of the records.

**Table 2.** Complexity

| Data Set Size | $k$-means | $k$-nearest neighbor | hybrid |
|---|---|---|---|
| 100 | 153 Sec | 177 Sec | 150 Sec |
| 1000 | 153 Sec | 900 Sec | 151 Sec |
| 9000 | 153 Sec | 7300 Sec | 172 Sec |

Table 2 presents a comparison of the classification time. We ran the classifiers on the same environment with the same data and similar configurations (cluster numbers). The classification was done using the LRU cache refinement. The results indicate that the $k$-nearest neighbors algorithm is by far the most time consuming. The time requirements of our hybrid algorithm are almost as low as those of the $k$-means algorithm, which is the most efficient.

*Realtime evaluation.* We tested the feasibility of our algorithm in a realtime embedded environment, by implementing the $k$-nearest neighbors and hybrid algorithms on the SCE2020 platform, one of Cisco's network traffic controllers. More specifically, the algorithms were developed and implemented as a standalone component (in C++, on a PPC dual core) on SCE 2020. We tested the accuracy of the algorithms by injecting the records (flows) using the Endace tool [6] and comparing the classification results.

The Endace tool injected the traffic in line rate as it was recorded, while the SCE2020 was configured to send a report on each classified flow. The report contained the flow five-tuple and the assigned application. The accuracy results were similar to our off-line tests, as expected. We offer the following conclusions.

*Technical Limitations.* The algorithm employs basic mathematical calculations, mostly simple additions and multiplications. Implementing such an algorithm may be more challenging on a platform that does not support floating point primitives, although this difficulty is of course solvable in software.

*Memory.* The algorithm used 76 bytes per flow on the statistics collection phase and one Megabyte for the training set. Taking into account a concurrency level of a few thousand flows (in the classification phase) and the use of an LRU table, the classifier uses only 4-5Mb. This is a very low figure (for core classifiers) and fits our realtime low memory usage requirement.

*Performance.* Running the algorithm did not appear to exert any stress on the CPU. This is not surprising considering, by comparison, the amount of work required by a DPI classifier. However, one must keep in mind that the SCE2020 box runs many tasks besides the classification, and hence it is expected of the classifier not to load the CPUs. This should be tested further.

Summarizing, there appear to be no technical, memory or performance limitations in implementing our algorithm in a real-world professional classifier. The algorithm has some practical limitations, such as a somewhat high average classification point. For further discussion of usability issues see Sect. 5.

## 5 Discussion

*Conclusions and directions for future study.* This paper presents a statistical algorithm enhancing and complementing traditional classification methods. Its strength is in points where traditional methods are relatively weak, most importantly in handling encryption, but also in asymmetric routing and packet disordering. The proposed algorithm is shown to be fast and accurate, and has no limitations to implementing it in professional realtime embedded devices. Hence it can be implemented as a complementary method for dealing with encrypted and other problematic flows.

*Misclassification.* Flow classification is usually employed by traffic controllers to enforce some policy on the traffic flow. The result of imposing a wrong policy may significantly affect the user experience. Thus, in some cases it is better to classify flow as "unknown" than to misclassify it. We propose to label traffic

as unknown based on the notion of *homogeneous neighborhoods*. Namely, define the flow neighborhood ($k$-nearest neighbors, for $k > 1$) as *homogeneous* if all neighbors have the same label. Then classify a given flow as follows. In case its neighborhood is homogeneous, give it the neighborhood label; otherwise, mark it as "unknown". Our results show that this approach reduces misclassification levels but results in many more "unknown" flows. This tradeoff poses an interesting direction for further investigation.

*Combining DPI and statistical approaches.* Our algorithm has several advantages over traditional methods but still, the fastest and most accurate way to classify simple HTTP traffic is by using DPI, relying on a simple string signature. Yet, our algorithm can be used in situations where traditional methods fail. Indeed, the strengths of the statistical approach correspond to the weaknesses of DPI. For example, flows that were not identified by the traditional classifier (such as encrypted flows), and were labeled as 'unknown', may now be classified correctly using our algorithm, with little additional computational cost. Combining DPI and our proposed algorithm in such a way also allows a quick and efficient way to cope with new applications (one of the major drawbacks of DPI classification). Specifically, until a DPI signature is generated for the new application, our algorithm may give the customer a quick solution.

## References

1. Basher, N., Mahanti, A., Mahanti, A., Williamson, C.L., Arlitt, M.F.: A comparative analysis of web and peer-to-peer traffic. In: Proc. 17th WWW, pp. 287–296 (2008)
2. Bernaille, L., Teixeira, R., Salamatian, K.: Early application identification. In: Proc. ACM CoNEXT, p. 6 (2006)
3. BitTorrent. Tracker peer obfuscation, `http://bittorrent.org/beps/bep_0008.html`
4. Crotti, M., Dusi, M., Gringoli, F., Salgarelli, L.: Traffic classification through simple statistical fingerprinting. Computer Commun. Review 37(1), 5–16 (2007)
5. Dewes, C., Wichmann, A., Feldmann, A.: An analysis of Internet chat systems. In: Proc. 3rd ACM SIGCOMM Internet Measurement Conf. (IMC), pp. 51–64 (2003)
6. Endace. The dag tool, `http://www.endace.com/`
7. Este, A., Gringoli, F., Salgarelli, L.: Support Vector Machines for TCP traffic classification. Computer Networks 53(14), 2476–2490 (2009)
8. Gummadi, P.K., Dunn, R.J., Saroiu, S., Gribble, S.D., Levy, H.M., Zahorjan, J.: Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In: Proc. SOSP, pp. 314–329 (2003)
9. Karagiannis, T., Papagiannaki, K., Faloutsos, M.: BLINC: multilevel traffic classification in the dark. In: Proc. ACM SIGCOMM, pp. 229–240 (2005)
10. Kim, H., Claffy, K.C., Fomenkov, M., Barman, D., Faloutsos, M., Lee, K.-Y.: Internet traffic classification demystified: myths, caveats, and the best practices. In: Proc. ACM CoNEXT, p. 11 (2008)
11. Madhukar, A., Williamson, C.L.: A Longitudinal Study of P2P Traffic Classification. In: Proc. IEEE MASCOTS, pp. 179–188 (2006)

12. McGregor, A., Hall, M., Lorier, P., Brunskill, J.: Flow Clustering Using Machine Learning Techniques. In: Barakat, C., Pratt, I. (eds.) PAM 2004. LNCS, vol. 3015, pp. 205–214. Springer, Heidelberg (2004)
13. Moore, A.W., Zuev, D.: Internet traffic classification using bayesian analysis techniques. In: Proc. ACM SIGMETRICS, pp. 50–60 (2005)
14. Nguyen, T.T., Armitage, G.J.: A survey of techniques for internet traffic classification using machine learning. IEEE Comm. Surv. & Tutor. 10, 56–76 (2008)
15. Paxson, V.: Empirically derived analytic models of wide-area TCP connections. IEEE/ACM Trans. Networking 2(4), 316–336 (1994)
16. Roughan, M., Sen, S., Spatscheck, O., Duffield, N.G.: Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification. In: Proc. 4th ACM SIGCOMM Internet Measurement Conf. (IMC), pp. 135–148 (2004)
17. Sen, S., Spatscheck, O., Wang, D.: Accurate, scalable in-network identification of p2p traffic using application signatures. In: Proc. 13th WWW, pp. 512–521 (2004)
18. Zander, S., Nguyen, T.T., Armitage, G.J.: Automated Traffic Classification and Application Identification using Machine Learning. In: Proc. 30th IEEE LCN, pp. 250–257 (2005)
19. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley, Chichester (2001)
20. Guyon, I., Elisseeff, A.: An Introduction to Variable and Feature Selection. J. Machine Learning Research 3, 1157–1182 (2003)