<http://www.diva-portal.org>

Postprint

This is the accepted version of a paper presented at *32nd International Conference on Advanced Information Networking and Applications (AINA)*. Krakow, Poland, 16-18 May 2018..

Citation for the original published paper:

Garcia, J., Korhonen, T., Andersson, R., Västlund, F. (2018)
Towards Video Flow Classification at a Million Encrypted Flows Per Second
In: Leonard Barolli, Makoto Takizawa, Tomoya Enokido, Marek R. Ogiela, Lidia Ogiela & Nadeem Javaid (ed.), *Proceedings of 32nd International Conference on Advanced Information Networking and Applications (AINA)* Krakow: IEEE
Advanced Information Networking and Applications
<https://doi.org/10.1109/AINA.2018.00061>

N.B. When citing this work, cite the original published paper.

© 2018 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:kau:diva-68705>

Towards Video Flow Classification at a Million Encrypted Flows Per Second

Johan Garcia, Topi Korhonen, Ricky Andersson, Filip Västlund

Department of Mathematics and Computer Science

Karlstad University, Sweden

Email: `firstname.lastname@kau.se`

Abstract—As end-to-end encryption on the Internet is becoming more prevalent, techniques such as deep packet inspection (DPI) can no longer be expected to be able to classify traffic. In many cellular networks a large fraction of all traffic is video traffic, and being able to divide flows in the network into video and non-video can provide considerable traffic engineering benefits. In this study we examine machine learning based flow classification using features that are available also for encrypted flows. Using a data set of several billion packets from a live cellular network we examine the obtainable classification performance for two different ensemble-based classifiers. Further, we contrast the classification performance of a statistical-based feature set with a less computationally demanding alternate feature set. To also examine the runtime aspects of the problem, we export the trained models and use a tailor-made C implementation to evaluate the runtime performance. The results quantify the trade-off between classification and runtime performance, and show that up to 1 million classifications per second can be achieved for a single core. Considering that only the subset of flows reaching some minimum flow length will need to be classified, the results are promising with regards to deployment also in scenarios with very high flow arrival rates.

I. INTRODUCTION

In many of today's cellular networks a considerable fraction of all flows, and a decisive majority of all packets, are from video related traffic. Video traffic places considerable demands on the network resources, and also plays an important role in user satisfaction. As a consequence, mobile operators often perform traffic management for this type of traffic in order to optimize their resource usage and provide the best possible quality of experience for their users. However, identifying particular flow types of interest, such as video, is becoming harder as end-to-end encryption becomes more pervasive on the Internet. Traditional deep packet inspection (DPI) methods are not applicable for encrypted traffic, so alternative mechanisms that do not require access to packet content are necessary.

In this work we examine the use of machine learning (ML) approaches to perform traffic classification based on flow characteristics. Such flow characteristics are available also for encrypted traffic and can be derived from information based on packet sizes, packet directions, and packet timings. Here we evaluate both the classification performance as well as the runtime performance of two ensemble-based machine learning methods, random forest and gradient boosted trees. To train the ML models and perform the evaluations we use a data set captured from a live cellular network consisting of more

than 2 billion packets transferred in over 40 million flows. The capture was performed by a specially instrumented DPI appliance that performs capture of per packet statistics in addition to using its DPI functionality to assign a ground-truth-like label to each flow. Our evaluation shows that a high classification performance (TPR ~ 0.95 , FPR ~ 0.05) can be achieved, and that the classification can be performed at a very high rate. Using a custom-built C-based random forest classifier we show that a classification rate of up to 1 million classifications per second can be achieved on a single core. Considering that a pre-filtering stage can remove a considerable fraction of nonrelevant very short flows, such as system may in typical traffic scenarios be able to scale to handling several million flows per second.

The remainder of the paper is structured as follows. Section 2 summarizes some of the related work in the field of flow classification while Section 3 describes the evaluation approach and gives an overview of the data set. Section 4 examines the classification performance from several viewpoints, and Section 5 covers the results from the runtime performance evaluation. Section 6 discusses the results followed by the conclusions in Section 7.

II. RELATED WORK

Related problems have been studied by a considerable body of previous works. Bernaille et al. [1] is one of the first to suggest using an initial subset of packets and apply ML for traffic classification. More recently, Alshammari et al. [2] presents a machine learning approach for identifying VoIP in encrypted network traffic. In the paper they compare three different supervised learning algorithms, C5.0, AdaBoost and Genetic Programming (GP). The reasoning behind the choices of algorithms were that all three use memory very well and do not have the drawbacks of significant memory overheads like for example Support Vector Machines (SVM). Two criteria were used to measure the performance of the three algorithms, detection rate (DR) and false positive rate (FPR). The results of the study were that C5.0 achieved the best performance of the three algorithms, with as high as 99% DR and 1% FPR for SKYPE while AdaBoost got 74% DR and 3% FPR and GP got 95% DR and 10% FPR for the same dataset.

In relation to cellular traffic classification, Taylor et al. [3] created a methodology and a framework called AppScanner for automatic fingerprinting and real-time identification of

Android application in encrypted network traffic. Traffic flows from apps are separated in interactive and non-interactive traffic, interactive traffic is traffic generated by user interaction while non-interactive traffic is generated without user interaction, such as when an application polls a server for updates. In the paper the methodology focuses primarily on interactive traffic. The network flows were used to derive statistical properties such as variance and skew of packets sizes in the different flows. There were two algorithms compared in the paper, Random Forest and SVM. Of the two algorithms random forest had the best performance, with the best results being 96% precision and 82.5% recall.

Peng et al. [4] focuses on the evaluation of statistical features. They consider six different feature sets covering payload size data, pure statistical features and a hybrid feature set. The feature sets are evaluated over three data sets using 10 different machine learning classification approaches. The results indicate that a small number of statistical features can provide strong classification performance. Xu et al. [5] examines how to select optimal features for use in statistical-based traffic classification, including the use of higher order statistical moments such as skew and kurtosis. Shbair et al. [6] consider classification of encrypted traffic, and suggest a multi-level approach to identify application classes in HTTPS traffic. Fu et al. [7] study encrypted cellular traffic, focusing in particular on how to classify mobile messaging apps. Correlating HTTPS traffic and DNS requests to allow host-based service classification of encrypted traffic is examined by Mori et al. [8]. Work by Casas et al. [9] uses DPI-labeled flows as ground truth with a semi-supervised learning approach for flow classification.

Bar-Yanai et al. [10] consider real-time classification of encrypted network traffic. The paper introduces a hybrid statistical algorithm which integrates both k-nearest neighbor (k-NN) and k-means algorithms. The algorithm is then tested on HTTP, Bit-torrent, SMTP, and EDonkey traffic. The k-means algorithm has a lower accuracy rate but requires significantly less computational power and is faster than k-NN. Because short flows (<15 packets) have unreliable statistical properties they were removed and only longer flows were used. It is mentioned that short flows account for 87% total flows but only 7% of total bytes of overall bandwidth. The results showed that k-means alone only averaged an accuracy rate of 83% and k-NN 99.1%. Together the used hybrid algorithm averaged a similar detection rate as k-means while the time complexity problem was reduced compared to the original k-NN algorithm.

Markov Chain Fingerprinting has been used to classify encrypted traffic as shown in the paper by Korczyński et al. [11]. The information in SSL/TLS headers were used to find statistical features on the traffic of some applications. A sequence of SSL/TLS messages in one direction can be modeled by a Markov chain. Traffic to and from twelve applications were studied in this paper. Some examples of the features of the applications were given in the paper; for example, 55% of new SSL/TLS sessions Twitter are resumed from previous sessions. Paypal on the other hand had 92.8% of the sessions start with a "Server Hello" message. Strong

classification results were observed and most applications had a TPR rate over 90% which varied a bit between the application and dataset, Amazon S3 for example had a true positive rate of over 97% regardless on what dataset was used for training and for validation while Dropbox was as low as 0.69% TPR for one of the sets.

Surveys in the area include Nguyen et al. [12] who survey early ML based classification approaches, and Finsterbush et al. [13] who provide a more recent survey of payload-based classification approaches. They consider several of the most used payload-based classification tools such as nDPI, Libprotoident and L7-filter. Using a testbed they evaluate both classification and runtime aspects for these classifiers. Velan et al. [14] survey existing approaches for classification of encrypted flows along the categories supervised learning, semi-supervised learning, basic statistical and hybrid methods.

There are no conclusive results which shows what feature-based traffic classifiers has the best results. A major factor hampering the ability to perform a comparative analysis is that the reported results to a very high degree depend on the datasets used, the particular classification task examined, and the parameterization of the considered models.

Differentiating factors for this work in relation to previous work are the use of a recent data set captured from a live cellular network, and that it implements an efficient C-based tree ML model parser and thoroughly considers runtime performance aspects.

III. DATA SET OVERVIEW

A. Data collection

The data set was collected from inside the cellular network backbone of a commercial cellular operator. An operational DPI box was modified to collect anonymized information for each packet in a flow, up to a maximum collection time of one minute per flow. For each packet, the data also included the flow application label as inferred by the DPI engine at that point in time. The DPI engine has over 1000 applications that it differentiates between, and it can update the classification of a flow as more packets are observed. A list of applications that were considered video were provided by the DPI vendor. Data collection was performed during 18 hours in February 2017. The data set consists of 42 million unique flows, which in the captured data set had a total of 2.1 billion packets and represented a transfer of 1.66 TiB.

Data sanitization was employed to remove flows which were started during the last 30 seconds of the capture window, or were initiated before the start of the capture window. As the capture was performed on a single link of a load-balanced connection pair where non-synchronized hashing was used to distribute the load, a large number of flows were only observable in a single direction. Flows with traffic observed in only one direction were removed during sanitization. After sanitization, 10.0 million flows and 824 million packets remained. The CDF of the flow sizes in the sanitized data set is shown in Figure 1. As indicated in the figure, the median flow size was only 13 packets and the distribution of flow lengths is heavily skewed towards short flows.

Figure 1. Flow length CDFs

Table I
DPI-LABELS FOR SANITIZED DATA SET (VIDEO CLASSES ARE BOLDFACE)

	DPI-label	Nr of flows	Mean DL packetsize	Observed packets	Transport protocol
1	DNS	2983 K	168	6.87 M	TCP/UDP
2	SSL v3	994 K	973	52.2 M	TCP
3	HTTP	781 K	1130	41.8 M	TCP
4	Google	753 K	923	48.7 M	TCP/UDP
5	Facebook	493 K	1021	46.7 M	TCP
6	YouTube	327 K	1378	255 M	TCP/UDP
7	Instagram	309 K	1323	87.4 M	TCP
8	Outlook.com	235 K	584	7.85 M	TCP
9	Apple	223 K	820	10.1 M	TCP
10	Bittorr. KRPC	163 K	653	1.35 M	UDP

The division of the flows among the different application DPI-labels in the sanitized data set is shown in Table I. Here we note that besides DNS, the most frequent DPI label is SSLv3 which signifies encrypted traffic that could not be assigned to a particular application class by the DPI engine. As it cannot be inferred which of these encrypted flows actually contains video traffic or not, flows with this label (or similar labels signifying undecidable flows) are not included when constructing the filtered data set used for training and subsequent evaluation of the machine learning models. Also clear from the table is that a very large fraction of data transferred in the network is video traffic. More packets are observed for the fifth-ranked YouTube label than for the total of the four most frequent flow labels.

B. Filtered data set characterization

Flow classification is often performed by using a subset of the initial packets in a flow. In this work we perform the classification based on features that are computed from the first 20 packets of a flow. Consequently flows with a length of less than 20 packets are not of interest here, and these short flows have been removed when creating the filtered data set used in the further examinations. The filtered data set now contains 3.75 million flows and 783 million packets, with the flow length distribution shown in Figure 1. Here we can note that the median of the filtered data set is 34 packets, and that the filtering has removed 63% of all flows from the sanitized set.

The most frequent video and non-video DPI labels in the filtered data set are listed in Table II. This table provides the mean downlink packet size and number of observed packets considering *only* the first 20 packets in each flow. It can

Table II
VIDEO AND NON-VIDEO DPI-LABELS FOR FILTERED DATA SET.
PACKETSIZE AND PACKET COUNT FOR FIRST 20 PACKETS IN FLOW.

	Video application	Nr of flows	Mean DL packetsize	Observed packets	Transport protocol
1	YouTube	229 K	785	4.58 M	UDP/TCP
2	HTTP media str.	34 K	1150	682 K	TCP
3	Netflix	16 K	1150	325 K	TCP
4	Kodi	583	1089	11.6 K	TCP
5	Flash video	447	1155	8.94 K	TCP

	Non-video application	Nr of flows	Mean DL packetsize	Observed packets	Transport protocol
1	Google	517 K	578	10.3 M	UDP/TCP
2	Facebook	320 K	459	6.40 M	TCP
3	Instagram	285 K	680	5.70 M	TCP
4	HTTP	261 K	916	5.21 M	TCP
5	Outlook.com	189 K	626	3.78 M	TCP

be noted that as a general trend the video applications have larger mean downlink packet size compared to the non-video applications, and this corresponds to one of many features used to construct a classifier.

IV. CLASSIFICATION PERFORMANCE EVALUATION

The data set is used to evaluate classification performance using the `scikit-learn` package [15]. As runtime considerations are important the focus is on forest based approaches over other approaches such as support vector machines (SVM).

A. Evaluated feature sets

The captured data set contains per packet traffic data in the form of packet arrival times, packet sizes, and packet directions. Moreover, the DPI engine service labels and video/non-video labels are available enabling supervised learning to be used. To extract flow related features from the packet data we consider various properties of a given flow of packets, however, restricting our consideration to the leading twenty packets of each flow thus focusing on early flow classification.

We generate two separate flow feature sets: (i) the 'statistical' (st) feature set contains features related to statistical properties of the packet flow, while (ii) the 'composite' (cp) feature set is more focused on the feature extraction speed from the packet traffic. The statistical feature set contains features that mostly focus on the statistical properties of the packet sizes of the leading packets. Thus, the feature set includes the packet sizes mean, standard deviation, variance, skew, and kurtosis, for each direction and overall. All the used features are labeled in Table III by 'st'.

Some of the statistical features are computationally rather expensive to extract and thus we introduce the composite feature set. This feature set only includes the mean of packet sizes but leaves the more complex statistical features out, however, replacing them with several other features. The features in the composite feature set require minimal computational effort, making this feature set favored when runtime is of concern. Features of composite feature set are presented in Table III with label 'cp'.

Table III

PER-FLOW FEATURES FOR THE STATISTICAL (ST) AND COMPOSITE (CP) FEATURE SETS. (U/D = UPLINK/DOWNLINK DIRECTIONS)

Feature label	Description	Use
nbt	Total amount of Bytes in the considered packets	st, cp
npu, npd	Number of packets in u/d	st, cp
mean_ps	Mean of packet sizes	st, cp
mean_psu, mean_psd	Mean of packet sizes u/d	st, cp
std_ps	Standard deviation of packet sizes	st
std_psu, std_psd	Standard deviations of packet sizes u/d	st
var_ps	Variance of packet sizes	st
var_psu, var_psd	Variance of packet sizes u/d	st
skew_ps	Skew of packet sizes	st
skew_psu, skew_psd	Skew of packet sizes u/d	st
kurt_ps	Kurtosis of packet sizes	st
kurt_psu, kurt_psd	The kurtosis of packet sizes u/d	st
fd	Flow duration, i.e., time between first and last packet	st, cp
nbu, nbd	Number of Bytes u/d	cp
nnfp	Number of non full packets, i.e., packets with less than 1400 Bytes	cp
mean_psdfrac	Mean packet size distance from running average. That is, the mean of values obtained by subtracting the current packet size running average from packet size.	cp
mean_pidfrac	Mean packet inter arrival time distance from running average. That is, the mean of values obtained by subtracting the current inter arrival time running average from packet inter arrival time.	cp
max_ps	Size of largest packet	cp
max_psu, max_psd	Size of largest packet u/d	cp
mean/max/min_piu, mean/max/min_pid	Mean, maximum and minimum of packet interarrival times up and down directions	cp
fstb_0-200-400-800-1600-3200	Fraction of interarrival times between given time intervals	cp
fpb_0-500-1000-1400-1520	Fraction of packets between given size intervals	cp
fpb(u/d)_0-500-1000-1400-1520	Fraction of packets between given size intervals in up and down directions	cp

B. Evaluation metrics

We use precision, recall, ROC-AUC and F1 scores to evaluate the performance of a given classifier. The definitions for recall, precision and F1 are straightforward

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}},$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \text{TPR},$$

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

where TP are true positives, i.e., actual video classified as video, FP are false positives, i.e., non-video classified as video and FN are false negatives, i.e., actual video classified as non-video. Moreover, the ROC curve is defined by using true positive rate (TPR, which equals recall) and false positive rate (FPR)

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

where TN are true negatives, i.e. non-video classified as non-video. The ROC-curve is the TPR w.r.t. FPR as the classification threshold probability (p_c) is varied from 0 to 1. Here, for given classification probability p given by the classification

Table IV

CLASSIFIER PERFORMANCE METRICS. DEFAULT PARAMETERS: $\dots_d$, GRID SEARCH PARAMETERS: $\dots_g$

Feature set	Classifier	ROC AUC	precision, precision_op	recall, recall_op	F1
composite	rf _d	0.988	0.669, 0.671	0.931, 0.930	0.779
	rf _g	0.993	0.670, 0.642	0.953, 0.958	0.787
	gbm _d	0.941	0.375, 0.339	0.840, 0.863	0.519
	gbm _g	0.994	0.703, 0.676	0.958, 0.964	0.811
statistical	rf _d	0.986	0.682, 0.594	0.929, 0.947	0.787
	rf _g	0.991	0.677, 0.632	0.944, 0.949	0.788
	gbm _d	0.930	0.339, 0.316	0.819, 0.840	0.480
	gbm _g	0.992	0.687, 0.647	0.947, 0.957	0.796

model, the item is assigned to the negative class if and only if $p < p_c$. The ROC area-under-curve (AUC) metric is then the integral of the ROC curve.

C. Random Forest versus Gradient Boosting Machine

As a first step we evaluate the performance of the two forest based classification methods random forest (rf) and gradient boosting (gbm), using their default parameters. Then, the model parameters were grid-search optimized for the specific target metric, in this case ROC-AUC, and additional results collected. All performance results are obtained using a 5-fold cross validation. For training, a 50%/50% balanced subset of 360.000 flows are used. As seen in Table II there are some 280.000 video flows in the filtered data set, and such balancing allows the training to be performed using 180.000 of those video flows. Evaluation of the classification performance is done on a separate subset, which is not balanced, thus having the same class balance as is present on the monitored link at approximately 8% video flows. The grid-search uses a smaller subset of flows and employs less cross validation to reduce the required computation time.

Grid search was performed with the following parameters. For random forest: number of trees in the forest [50,75,100], max depth of a tree 14 to 41 in steps of 2, max number of features in a tree 4 to 16 (statistical) or 14 to 34 (composite) in steps of 2, and the criterion for best split either 'gini' or 'entropy'. The optimal parameters found were for statistical (composite) feature set: number of trees 100 (100), max depth 34 (22), max features 6 (26), criterion 'entropy' ('entropy').

For gradient boosting the number of trees were fixed at 300 and the search space was: learning rate .05 to .2 in steps of 0.05, max depth 4 to 18 in steps of 2 and max number of features 10 to 20 in steps of 5 (statistical) or 20 to 40 in steps of 10 (composite). The optimal parameters were for statistical (composite) feature set: learning rate 0.1 (0.1), max depth 14 (14), max features 10 (20).

The results with the default classifier settings are seen in Table IV in the rows with subscript $\dots_d$. Using the parameters found with the grid search, i.e., the ones maximizing ROC-AUC, yielded the results in the rows with subscript $\dots_g$.

The optimized classifier models perform consistently better for ROC-AUC, especially noticeable is the increase in performance of gbm as the parameters are optimized. With the default parameters gbm obtains precision values of ~ 0.3 , while after optimization precision is ~ 0.7 .

Figure 2. ROC and precision-recall curves with operating points.

Comparing the random forest (rf_g) and gradient boosted trees (gbm_g) classifier results as shown in Table IV, they appear fairly similar. To provide a more comprehensive view of their relative performance, a ROC curve is provided in Figure 2. The ROC-curve displays the relationship between the True Positive Rate (TPR) and the False Negative Rate (FNR), with an ideal location towards the top left corner. The chosen operating point is shown as a dot on the curves. Here, the operating point chosen is the point closest to the optimal position of the top left corner, and not the default operating point provided by the classification algorithms. Results for the chosen operating point is also listed in Table IV, as the second value in the precision and recall columns ($.._{op}$). Selecting the closest to optimal operating point does here not yield significant differences from the default point. In addition to the ROC-curve, Figure 2 also provides a precision-recall curve. Here we note that the trade-off between precision and recall can be performed by arbitrarily changing the operating point, i.e., the probability threshold p_t at which a given flow gets classified as video. By default this threshold is set to 0.5, however, lower threshold values increase the amount of flows classified as a video and thus increase the recall. With a decreasing probability threshold more flows that are non-video are unfortunately also being classified as video thus decreasing the precision of the classifier. As the flow data is unbalanced with only $\sim 8\%$ of flows expected to be labeled as video, even a fairly low FPR will lead to a low precision. This trade-off between precision and recall is presented in Figure 2, where precision is plotted against recall as the probability threshold p_t is varied from 0 to 1.

Based on the curves in Figure 2 it is possible to select an operating point on the curve that is appropriate for the particular use case at hand. For a use case where traffic management is applied to video flows to improve end-user QoE it might be tolerable with a fairly low precision. If the classification is tied to traffic type accounting, another operating point with higher precision but lower recall might instead be appropriate.

D. Statistical versus composite feature sets

The experimental data can also be examined from the viewpoint of feature set performance. Examining the results in Table IV and Figure 2 it can be seen that the composite features provide slightly better classification performance than

Figure 3. ROC-AUC as number of trees increases. Shaded area represents the deviation in cross validation runs.

the statistical feature. As we in this work are concerned with runtime performance this is encouraging as the composite features are less computationally demanding than the statistical ones. The features in the composite feature set can readily be implemented using counters and simple operations.

An analysis of the feature importance across the two feature sets showed that the variation in relative feature importance across individual features is larger for the composite than for the statistical features, and that several of the composite features are of negligible importance, whereas for the statistical feature set most of the features are utilized. There are more features in the composite feature set, and such large variation indicates that the feature set can potentially be further condensed. The unimportance of several features in the composite set is probably a consequence of the choice to consider only the twenty leading packets in the flows. That is, several of the features in the composite set would have higher predictive power if the number of examined packets were larger than the first 20. Experiments with larger number of considered packets and additional feature engineering aspects are however outside the scope of this paper.

E. Model complexity evaluation

The classification results discussed so far has been based on the model configuration values reported as providing the highest ROC-AUC values during grid search. We now extend the discussion to also consider the trade-off between classification performance and model complexity. The model complexity also has runtime implications which will be examined in the next section.

The first examination considers the case where model complexity, in terms of number of trees, is more widely evaluated relative to the boundaries set for the grid search. Results over an extended range are shown in Figure 3 for the two classifiers and feature sets. Here we can note that there is a pronounced knee in the graphs, after which an increase in the number of trees only provide minor improvements to the ROC-AUC. The results consistently shows the composite data set to provide better results than the statistical, regardless of the number of trees used. To provide a fuller picture of the complexity versus performance trade-off we also performed experiments for the composite feature set while jointly varying the maximum depth of the individual trees in the forest, and

Figure 4. ROC-AUC over model complexity for random forest classifier

Figure 5. ROC-AUC over model complexity for gradient boosting classifier

the number of trees. These results are shown in Figures 4 and 5. The results again show that only minimal performance increase can be achieved by increasing the number of trees above ~ 50 and the maximum depth above ~ 20 for random forest, and above ~ 12 and ~ 300 for gradient boosting.

V. RUNTIME PERFORMANCE

In addition to the classification performance examined in the previous section, runtime performance is of interest. In order to study runtime performance a specialized C-based tree evaluation implementation was developed. To be able to export trained scikit-learn models, a python export function was developed to represent the forest as a set of trees in a CSV file. For the runtime performance tests the CSV file is then read by a C program that sets up internal data structures to represent the forest, and then classifications is performed while measuring the required time. The performance is evaluated over 1 million flows and average classification time per flow is the metric of interest. It can be noted that the classification code is identical for both random forest and boosted trees. The difference between the two lies only in the structure of the forest generated during training. For the random forest algorithm the runtime characteristics were evaluated over an amount of trees that ranged from 10 to 80, and a maximum node depth between 10 and 40. Only every second value is tested. The gradient boosting algorithm was evaluated over a range of trees from 100 to 1000 in increments of 100, and every maximum node depth from 3 to 20.

A. Base case performance

The results for the base case is shown in Figures 6 and 7. The results in these figures were achieved when running on a single core of an i7-6850K CPU and using the GCC compiler with optimization settings -O3. Additional experiments were also performed with only baseline compiler settings without optimization and these were noticeably slower, especially for the gradient boosting case. In the remainder, results reflect the

use of -O3 optimization settings. From the graphs it is clearly evident that the random forest approach overall has much lower classification times than gradient boosting. It can also be seen that while random forest shows a smooth and regular surface as the parameters increase, the gradient boosting has a ridge line dependent on the maximum depth and number of trees. Coupling back to the classification performance evaluation in the previous section, we can observe that for appropriate classifier settings (i.e RF: MD= 20, trees=50, GBT: MD=12, trees=300), these baseline measurements indicate a classification time of around 4 μ s for in the forest and 15 μ s for gradient boosting trees. Given the big difference in runtime performance with no equivalent significant gain in classification performance, the following examination focuses solely on random forest.

B. Implementation optimizations

We now turn our attention to the internal code structure of the C program that performs the classification, with the aim to explore further ways to improve runtime performance. In the initial implementation, classification is performed on a flow by flow basis. Thus, a single flow is processed at a time, and the flow is evaluated against the forest on a tree by tree basis. The first optimization modifies how evaluation is performed. In this optimization we instead perform the evaluation on a batch of flows, and start with the first tree in the forest and evaluate this tree for all flows in the batch. This is then continued for the remaining trees in the forest. After the batch has been processed the classification results are available in an array. The classification times after this modification are shown in Figure 8. Comparing to Figure 6, it is clear that this optimization provides a considerable improvement in classification time.

The second optimization that was explored is based on the fact that forest-based classifiers employ majority voting. Consequently, it is possible to preempt the evaluation of the trees in the forest when one of the classes has passed the required majority threshold with the votes already cast by

Figure 6. Random forest baseline

Figure 8. Random forest with flow batching

Figure 7. Gradient boosting baseline. Note difference in scale.

Figure 9. Random forest with flow-batching and majority break

the thus far evaluated trees. The code was modified to test the number of votes after each tree in the forest has been processed. If any class has majority the remaining trees are not evaluated for that flow. Thus, at each iteration every flow is checked for potential preemption of the tree traversal for the current tree. The results for the combination of the first and second optimizations are shown in Figure 9. Also for this optimization a noticeable improvement in runtime performance was achieved.

Additional optimization attempts were also evaluated without providing significant performance gains. One such example is to modify the code to employ two evaluation loops with the first one utilized for the first half of the trees and which performs no checks for potential preemption as class majority cannot be achieved when less than half of the trees have been evaluated. Only in the second loop which is ran over the last half of the trees this additional check of potential majority is performed. The results however showed that this two loop approach did not provide improved performance over using a single loop. The optimization tests also showed that

the particular optimization outcome can vary between different instances of ML models, making it beneficial to consider runtime optimizations on a per model / per dataset basis.

VI. DISCUSSION

The data set characterization in Section 3 showed that a majority of all flows on the examined cellular backbone link has very short flow lengths. In such circumstances, a flow classification approach that uses more than a few packets will see a considerably smaller number of flows to be classified than the total amount of flows on the considered link. For the selected threshold examined here, 20 packets, the reduction in the number of flows is approximately 63%. To sustain a total incoming flow rate of 1 million flows per second, the classification rate would have to be $\sim 400,000$ flows per second. This gives a time budget of $2.5 \mu s$ to perform the classification, which can be sufficient as shown in the previous section. While there are much work on flow classification, few works in the literature consider the runtime aspects. Rizzi et al. [16] present a neuro-fuzzy classifier for which they report a classification

rate of below 200.000 flows per second on the evaluated FPGA hardware.

Also to be considered is the computational effort to obtain the feature values. Several of the features correspond to counters already present in an DPI implementation. Initial testing of the composite feature computation time indicates that they can be computed in a few microseconds on a single core. Consequently, the system presented here should be well-suited to handle millions of flows per second.

One important issue not considered in this work is the transferability of models. The ability to train a classification model on one link and deploy it on many (similar) links is desirable. However, the extent of model transferability could not be evaluated here as data was not available from multiple links. A related issue is model evolution over time which also would need additional data to evaluate. Another issue to consider is the semantics of the DPI labels and what is actually learned by the classifier. The DPI labels in many cases give an indication of which application was used to create the traffic, and are not necessarily reflecting a particular user activity and the resulting type of data transferred. As an example, the YouTube app can be used to browse among available videos, or to view the video. Both these actions would be labeled as YouTube by the DPI, but if the objective is to detect flows that perform actual video transfer such ground truth does not provide the optimal training. Utilizing unsupervised machine learning to improve the quality of the DPI training labels has been initially explored for a smaller data set in [17], and further elaborated for the current data set in [18].

VII. CONCLUSIONS

As end-to-end encryption becomes the norm for more and more Internet traffic, traditional approaches for traffic engineering based on DPI-type flow type identification can no longer be used. Here we utilize traffic that currently can be classified by a DPI engine to create a ML classifier. This classifier work on flow characteristics available also for encrypted traffic. In this study a large data set of real traffic has been examined both from the perspective of classification performance, as well as runtime performance. We focus on tree based machine learning approaches over slower alternatives such as SVM and others. In particular, the random forest and the gradient boosted tree approaches are thoroughly examined. The impact of the composition of the feature set is also examined. Specifically, the relative performance of a more compute intensive statistics-based feature set is contrasted to another feature set which has less computational demands. The results show that the use of less compute intensive features do not reduce the classification performance.

We further implement an efficient C-based classifier model implementation and evaluate the runtime performance. The results shows that the random forest approach has a considerable runtime performance benefit over gradient boosted trees while providing similar classification performance. The obtained performance figures indicate that a well implemented random forest approach is capable of sustaining classifications for an incoming flow rate of several million flows per second using only modest hardware resources.

ACKNOWLEDGMENTS

The authors wish to thank Sandvine for assisting with feature suggestions and data collection. Funding for this study was provided by the HITS project grant from the Swedish Knowledge Foundation. Computations were performed on resources at Chalmers Centre for Computational Science and Engineering (C3SE) provided by the Swedish National Infrastructure for Computing (SNIC).

REFERENCES

- [1] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamati, "Traffic classification on the fly," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 2, pp. 23–26, 2006.
- [2] R. Alshammari and A. N. Zincir-Heywood, "Identification of voip encrypted traffic using a machine learning approach," *Journal of King Saud University*, vol. 27, pp. 77–92, 2014.
- [3] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," *IEEE European Symposium on Security & Privacy*, vol. 16, pp. 439–454, 2016.
- [4] L. Peng, B. Yang, Y. Chen, and Z. Chen, "Effectiveness of statistical features for early stage internet traffic identification," *International Journal of Parallel Programming*, pp. 1–17, 2015.
- [5] M. Xu, W. Zhu, J. Xu, and N. Zheng, "Towards selecting optimal features for flow statistical based network traffic classification," in *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*, pp. 479–482, 2015.
- [6] W. M. Shbair, T. Chole, J. François, and I. Chrisment, "A Multi-Level Framework to Identify HTTPS Services," in *IEEE/IFIP Network Operations and Management Symposium*, pp. 240–248, 2016.
- [7] Y. Fu, H. Xiong, X. Lu, J. Yang, and C. Chen, "Service usage classification with encrypted internet traffic in mobile messaging apps," *IEEE Transactions on Mobile Computing*, vol. PP, no. 99, pp. 1–1, 2016.
- [8] T. Mori, T. Inoue, A. Shimoda, K. Sato, K. Ishibashi, and S. Goto, "Sfmap: Inferring services over encrypted web flows using dynamical domain name graphs," in *International Workshop on Traffic Monitoring and Analysis*, pp. 126–139, Springer, 2015.
- [9] P. Casas, J. Mazel, and P. Owezarski, "Minetrac: Mining flows for unsupervised analysis & semi-supervised classification," in *2011 23rd International Teletraffic Congress (ITC)*, pp. 87–94, Sept 2011.
- [10] R. Bar-Yanai, M. Langberg, D. Peleg, and L. Roditty, "Realtime classification for encrypted traffic," in *International Symposium on Experimental Algorithms*, pp. 373–385, Springer, 2010.
- [11] M. Korczyński and A. Duda, "Markov chain fingerprinting to classify encrypted traffic," in *Infocom, 2014 Proceedings IEEE*, pp. 781–789, IEEE, 2014.
- [12] T. 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.
- [13] M. Finsterbusch, C. Richter, E. Rocha, J.-A. Muller, and K. Hanssgen, "A survey of payload-based traffic classification approaches," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 1135–1156, 2014.
- [14] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, "A survey of methods for encrypted traffic classification and analysis," *International Journal of Network Management*, vol. 25, no. 5, pp. 355–374, 2015.
- [15] 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.
- [16] A. Rizzi, A. Iacovazzi, A. Baiocchi, and S. Colabrese, "A low complexity real-time internet traffic flows neuro-fuzzy classifier," *Computer Networks*, vol. 91, pp. 752–771, 2015.
- [17] J. Garcia, "A clustering-based analysis of DPI-labeled video flow characteristics in cellular networks," in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM) AnNet Workshop*, IEEE, 2017.
- [18] J. Garcia and A. Brustrom, "Clustering-based separation of media transfers in dpi-classified cellular video and voip traffic," in *IEEE Wireless Communications and Networking Conference (WCNC)*, IEEE, 2018.