

A Novel Flow-level Session Descriptor With Application to OS and Browser Identification

Omar Richardson

Dept. of Mathematics and Computer Science
Karlstad University
Karlstad, Sweden
Email: omar.richardson@kau.se

Johan Garcia

Dept. of Mathematics and Computer Science
Karlstad University
Karlstad, Sweden
Email: johan.garcia@kau.se

Abstract—High level traffic characteristics have the potential to be useful for inference of various host characteristics. This work proposes the novel Flow-Discretize Order (FDO) approach for describing session characteristics in an intuitive manner, while also retaining flow ordering information. The FDO approach allows for flexible construction of flow descriptors, by using different flow properties and applying appropriate discretization. The individual flow descriptors are concatenated to form session descriptors. By utilizing string distance metrics, such as the Damerau-Levenshtein distance (DLD), it is possible to perform both unsupervised and supervised learning on the FDO session descriptors. Here, we utilize FDO as a tool for OS and browser identification coupled to a particular user activity, in this case watching YouTube videos. The variable-length nature of FDO session descriptors precludes learning methods expecting fixed dimensionality from being used. However, experiments show that excellent performance are provided by methods operating on distances such as hierarchical Ward for the unsupervised case, and k-NN for the supervised case. The supervised learning evaluation shows that over 99% accuracy can be achieved for both operating system and browser identification based on video session characteristics. The FDO framework also provides multiple promising avenues for further research and improvements such as improved methods for discretization boundary placement, more elaborate feature selection approaches, and more fine-grained DLD weights.

I. INTRODUCTION

Learning from network traffic can be performed at multiple levels. Packet data content, packet header content, packet timings, network flow characteristics, as well as session characteristics created by groups of flows, can all be analyzed to infer relevant information. In this work we propose a novel Flow-Discretize Order (FDO) method using flow-level information to produce session descriptors. These session descriptors can be used to perform inference of properties such as operating system (OS) or browser employed at hosts in the network. One context for browser and OS fingerprinting is network managers who wish to obtain statistics regarding OS and browser usage among users in their network for operational considerations [2], [3]. In a security context, there can be benefits from fusing information from intrusion detection systems (IDS) with OS identification information to determine which hosts are susceptible to attacks [5], [15].

Most previous work on OS identification relies on the ability to inspect headers or content in individual packets. The FDO approach instead relies on quantized flow descriptors that are concatenated to variable length session descriptors. Using these session descriptors, we explore several unsupervised learning methods for grouping end-user characteristics such as OS and browser for a particular user activity, in this case watching YouTube. As the session descriptors are of varying dimensionality, appropriate learning methods are necessary that can handle such a constraint. As traditionally used distance metrics such as Euclidean and Manhattan [22] are not applicable for varying length descriptors, distance metrics coming from the field of string matching are instead adapted and optimized. Supervised learning is also employed with the session descriptors, and show that FDO is capable of identifying OS and browser classes with an accuracy over 99%, even for small numbers of training instances. The FDO approach is novel and flexible, and provides several avenues for future work.

This paper is structured as follows. In Section II we outline related contributions, while Section III describes our proposed Flow-Discretize Order (FDO) method. Section IV describes the data collection and some data set characteristics. In Section V and Section VI, unsupervised and supervised learning strategies are applied and the results analyzed. Finally, Section VII and Section VIII discuss the proposed techniques, and provide conclusions.

II. RELATED WORK

The issue of operating system identification has been the subject of much interest in the security domain. Active OS fingerprinting is a technique that actively sends packets to the target host with the aim to generate replies which can be examined in order to infer the operating system of the target host. Examples of such approaches are provided in [6], [32], [33]. More related to the approach discussed in this paper are the passive OS classification approaches. These approaches examine packets such as TCP SYNs and HTTP requests to infer the OS used but without provoking any additional traffic for the examined host. One such approach is discussed by Matoušek et al. [26]. Aksoy et al. [2] examines the ability to

perform OS classification based on single ICMP, UDP or TCP packets using different classification techniques and a tailored feature selection approach. Anderson et al. [5] also consider a passive approach, but additionally process multiple flows from the same user to improve classification performance and provide detailed OS information. However, in contrast to our proposed approach [5] considers TCP packet headers as well as TLS and HTTP content data. Hybrid approaches using both passive and active techniques for OS discovery have also been proposed [9], [15].

With regards to browser identification, an extreme case is an attacker that controls a website and who wishes to determine not only which browser is used, but also as much additional information as possible to limit browsing anonymity. An early work in this direction was performed by Eckersley [12], which showed that over 90% of browsers with Flash or Java were in fact uniquely identifiable from a set of more than 280000 browsers. Recent work by Gomez-Boix et al. indicate that the anonymity has been somewhat improved as only 34% of 2 million browser fingerprints were unique in their study [19]. Such user/browser identification approaches are of an active nature and generate considerable traffic to perform probing of browser component characteristics, and can be seen as tangential to the aim of passive browser identification.

An approach for browser identification based on flow-level information rather than per packet data is proposed by Yen et al. [39]. Using a set of nine main features based on flow and session statistics they report a precision of 71% as well as a recall of 71% with an SVM classifier over two browsers. Their experiments also highlight that the ability to identify the browser increases the ability of an attacker to classify which website a user is visiting based on flow level information. Similarly, Alan et al. [4] show that browser diversity is sufficiently large to allow browser-specific classification to improve performance for a web fingerprinting task.

Previous work in the literature which considers the joint detection of browser and operating system include work by Muehlstein et al. [27] who for a proprietary data set report a operating system classification accuracy of 100% over three OSs and a browser classification accuracy of 99% over four browsers. They utilize an SVM-RBF classifier but, in contrast to the present work, they utilize features computed on a per packet basis rather than only flow level information.

III. THE FLOW-DISCRETIZE ORDER METHOD

A. FDO overview

A high-level view of the FDO method is provided in Figure 1, which shows the operations performed to obtain the session descriptors in the left part. The right part shows the operations for unsupervised clustering and k -NN based classification.

In short, the FDO method first computes descriptive features of the flows making up a session, discretizes these features, and concatenates them to build a flow-order dependent session descriptor. Effectively, the session descriptor captures several aggregate aspects of the behavior of a particular session. This

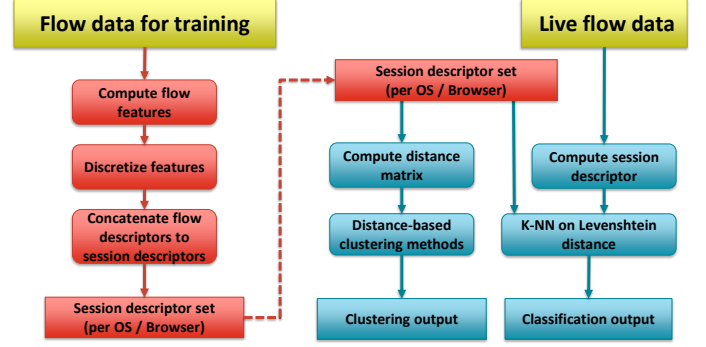


Fig. 1: Overview of Flow-Discretize Order (FDO) approach.

TABLE I: Flow features. (U) Only uplink, (D): Only downlink, (T): Both directions

Flow feature	Unit	Flow descriptor specification		
		W_{standard}	W_{burst}	$W_{\text{direction}}$
Flow size T/U/D	bytes	2 (D)	2 (D)	2 (D)
Flow size T/U/D	packets	0	0	0
Relative start time	seconds	2	2	2
Majority is downlink	-	0	0	1
Flow lifetime	seconds	1	0	0
Is Bursty	-	0	1 (D)	0
TCP / UDP	-	1	1	1
Nr concurrent flows	<count>	0	0	0

session behavior will be different for different activities, but may be also different for different software or operating system implementations when performing a particular activity. Thus, we hypothesise that the session descriptors are useful for inference with regards to which OS and browser is used to perform a particular activity.

Another potential use-case for the FDO method is to learn structure from the traffic, i.e. find clusters of behavior that represent a systematic behavior displayed by a set of users. After performing session level clustering with FDO, the identified clusters can then be analyzed further to infer understanding of what the identified clusters represent.

B. FDO flow descriptors

The first step in FDO is the computation of a flow descriptor composed of information from multiple flow features. A listing of flow features are shown in Table I, which also indicates the number of bits assigned to each feature for three different flow descriptor specifications. Since the main use case of FDO encompasses situations when per-packet information is not present, the first descriptor considers a set of features which employs only per flow information, as would be available by for instance NetFlow [10] or IPFIX [35] records. The features are discretized according to a flow descriptor specification, W , creating a *standard* flow descriptor. For comparison, we also construct a *burst* flow descriptor which additionally uses per packet information to derive whether a flow is bursty or not. Bursty behavior is common in video streaming applications as the transfer of video data is performed in bursts to fill

up a play-out buffer at the client. Finally, a *direction-based* flow descriptor is devised as well, where one of the features indicates if the majority of the payload is sent in downlink direction. For the considered flow descriptor specifications manual heuristics were used to select the features and bit allocation per feature. We note that the FDO approach can readily be extended by considering more elaborate schemes for such selection and allocation.

From a more formal perspective, we consider a flow as a sequence of packets that share the properties source address, source port, destination address, destination port and transport layer protocol. Denoting this 5-tuple of properties of packet p out of packet stream P as $f(p)$, we can define a flow c as follows:

$$c := [p_i \in P | f(p_i) = a], \quad (1)$$

where a is a given 5-tuple. For each flow a monitoring process provides, or allows the derivation of, a number of flow features as listed in Table I. By representing a flow with such features we can obtain a flow representation in an n -dimensional space, which can subsequently be binned into a m -bit representation.

Let $g_1, \dots, g_n$ be functions that map the flow to the desired n features. Our non-discretized flow representation becomes

$$G(c) := [g_1(c), \dots, g_n(c)]^T. \quad (2)$$

Next, we discretize the flow representation $G(c)$ with a discretization specification $W = (w_1, \dots, w_n)$ (see Table I). This vector associates every feature g_i with a number of bits w_i for any $i = 1, \dots, n$ such that $w_1 + \dots + w_n = m$. To find the proper bins for each flow property, we pick a binning function b_i that discretizes every value $g_i(c)$ into one of 2^{w_i} bins.

For ease of notation, let us denote $W_i := w_1 + \dots + w_i$. Then, the discretization r of flow representation $G(c)$ using discretization specification W can be computed as follows:

$$r(c) := b_1(g_1(c))2^0 + b_2(g_2(c))2^{W_1} + \dots + b_n(g_n(c))2^{W_{n-1}} \quad (3)$$

This flow descriptor $r(c)$ can be uniquely represented by a letter of a m -bit alphabet. There are several possible binning functions b , see [31] for a survey. For our experiments we choose the equal frequency binning function for b , and set $m = 6$ bits.

C. FDO session descriptor construction

Next, a session descriptor is constructed as a concatenation of all flow descriptors in a session. More precisely, for a session with flows $c_1, \dots, c_N$, we define the corresponding session descriptor

$$s := r(c_1) \cdot r(c_2) \cdot \dots \cdot r(c_N), \quad (4)$$

where $a \cdot b$ represents the concatenation of characters a and b . This way, we obtain a string-based representation of the session, which we use to classify different configurations with. A few examples of descriptors for sessions with different configurations are presented in Table II.

Note that different sessions with the same OS and browser configuration are likely to have different session descriptors.

Due to the volatile behavior of how flows are initiated and used for data transfer, not only can different flows arrive in different orders, it is also possible for them to have different flow descriptors. For this reason, we require a distance metric which is flexible with respect to reordering and substitution, as well as insertion and deletion of characters.

D. FDO session descriptor distances

The Levenshtein distance [23] of two strings s_1 and s_2 is a metric that can be interpreted as the minimum number of edits (insertion, deletion, substitution) on characters of s_1 required to obtain s_2 . Numerous variants of string distances exist [1], [28], [30]. In particular, [11] extended the Levenshtein distance to also account for the transposition of two characters; this extension is also called the Damerau-Levenshtein distance (DLD). We introduce three edit operations: f_P (insert/deletion), f_T (transposition) and f_S (substitution). Any two strings defined over the same alphabet can be obtained by repeatedly applying these operators. An edit sequence $E = [f_1, \dots, f_l]$ applied from s_1 to s_2 is valid if the following holds:

$$E(s_1) := (f_1 \circ f_2 \circ \dots \circ f_l)(s_1) = s_2. \quad (5)$$

Let $N_P(E)$, $N_S(E)$, $N_T(E)$ be respectively the number of f_P , f_S and f_T operations in E . We define the weighted DLD d_L between s_1 and s_2 as follows:

$$d_L(s_1, s_2) = \min_{E \in \mathcal{E}} c_P N_P(E) + c_S N_S(E) + c_T N_T(E), \quad (6)$$

where $\mathcal{E}$ represents the set of all valid edit sequences. We assign the edit costs in the classification evaluation as $(c_P, c_S, c_T) = (0.5, 1.0, 0.1)$, based on a grid search as discussed in a later section. Compared to substitution operations, insertion and deletion have a low cost to account for the volatility of flows: even among sessions with identical configurations, differences occur in flows carrying non-video data. Transpose operations have a lower cost assigned to them to account for the lack of a strict time-ordering of the flows.

IV. EVALUATION PRELIMINARIES

A. Data collection

The data for this experiment is collected by capturing outgoing and incoming traffic for an instrumented client machine scripted to retrieve streaming content from the youtube.com website over multiple sessions. To examine a variety of browsing conditions, the client machine was connected to the internet via an emulation machine which could provide bandwidth throttling. Four different settings were used: no throttling, 8, 5, and 2 Mbps. The client browser was further configured to extend over eight different view port sizes, from 256x144 to 3840x2160 pixels. Four different videos were considered (from 293 to 1197 seconds long), leading to 128 different combinations. Further, two different browsers (Chrome and Firefox) and two different operating systems (Windows 10 and Ubuntu Linux 16.04) were utilized. Data was collected as a pcap file, and post-processed to get per-packet timing and size data. The packet-level data was used to

TABLE II: Session descriptor examples.

Configuration:	Session descriptor
Windows, 5 Mbps, Firefox, 1920x1080	IIIIYIQIIIIQIIIIYII2I2II2II2II2IYQII0QYIQYI0IIIIYIQQ2IQII2IIIIQ22II2... (len 177)
Windows, 2 Mbps, Chrome, 426x240	GOGGGGOGG2222YY22222222YYQY2Y2YY20IYQY2ZJZJ3ZRRJZ3JZ3JJZZ (len 57)
Linux, no shaping, Firefox, 1920x1080	001YY2YY2QY0022222YQY200ZZR3R=RRZ3JV=R3RJ333RVZ==R=JRRRRR=R (len 60)
Linux, no shaping, Chrome, 1280x720	00XYYYYY2YQ0022222QY02YYY22JVZ3R=VZ3=RRRVJRJVVR333VVRZRVJRJVVRVR... (len 82)

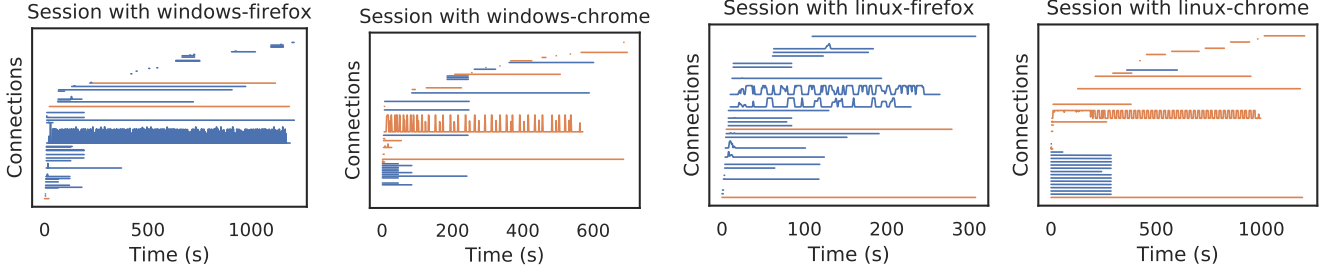


Fig. 2: Connection structure within four example sessions. Blue: TCP, Orange: UDP.

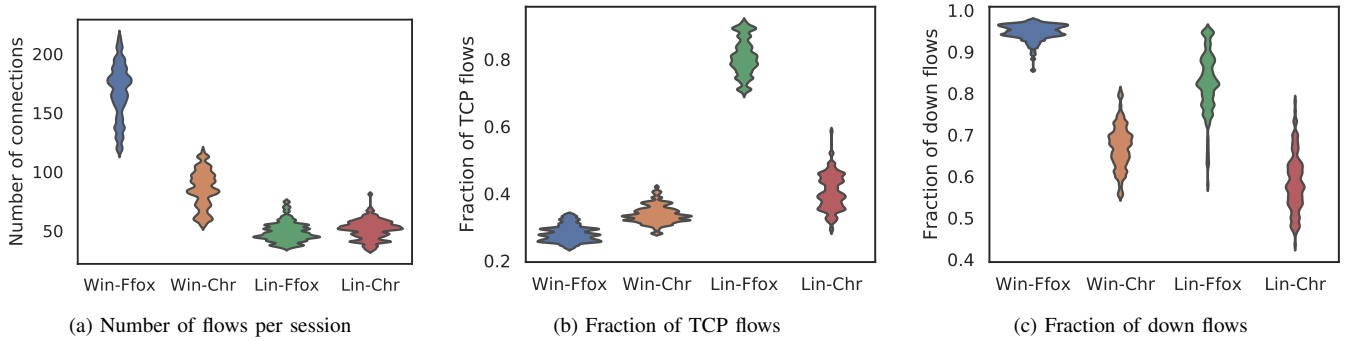


Fig. 3: Session characteristics, grouped by the four considered class combinations.

generate the “Is Bursty” feature. All other features considered only flow-level information, similar to what is provided in basic NetFlow data. It should be noted that no packet content is inspected, and FDO is oblivious to any encryption of the flow content.

B. Overview of session and flow characteristics

Figure 2 provides four visual examples of how sessions are composed of many TCP and UDP flows spread over time. Additionally, there are a number of very short flows which are too small to be drawn but which can be inferred from sections of vertical white space. The four examples in Figure 2 can serve as a snapshot, but statistics over the population are necessary to gain more insights. The distribution of three session characteristics are shown in Figure 3 as violin plots which display the kernel density estimated [36] vertical PDFs with a Gaussian kernel, $\text{bw}=0.1\sigma$. From the session characteristics, it is evident that there are differences between the different OS and browser configurations. These features, each taken in isolation, do however not allow for complete class separation. As a visual illustration of how these differences manifest with regard to the FDO session descriptor, Figure 4 shows the distance matrix between all sessions, ordered first

by operating system and secondly by browser, using costs as defined in the previous section. The four distinct blocks in each direction suggest that the session descriptors are capturing relevant aspects for OS and browser separation, although the variation in pairwise distances within the blocks indicate that other properties affect the DLD as well.

V. UNSUPERVISED LEARNING EVALUATION

This section considers an unsupervised learning approach to derive structure regarding client configurations from the session descriptors. In the unsupervised learning process, the precomputed DLD between the session descriptor instances is used to group the sessions into clusters. Note that the only applicable clustering algorithms are the ones that do not require the localization of the flows in an n -dimensional feature space; the edit distance framework does not provide us with such a structure. Because only pairwise distances are available, objects like centroids are not well defined, eliminating a range of otherwise usable clustering methods. However, connectivity-based, density-based and similarity-based clustering methods are still applicable. To evaluate the performance of unsupervised learning four different clustering approaches were examined.

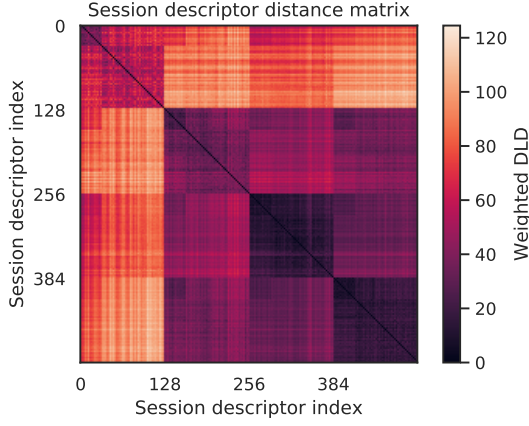


Fig. 4: Pairwise distances between all sessions when ordered by OS and browser, using W^{standard} and costs as defined in Section III-D.

A. Evaluated clustering algorithms

DBSCAN: Density-based spatial clustering of applications with noise (DBSCAN) [13] is a clustering approach that constructs clusters based on similar spatial densities. Clustering is done based on a neighborhood size parameter ε and an instance number threshold m . The algorithm starts by identifying a *core* instance: an instance that has at least m instances at a distance of at most ε . This core instance will form a new cluster. This cluster is recursively expanded with other instances having a distance less than ε to any instance in the cluster. When no more such instances are present, the algorithm continues by identifying a previously unclustered core instance, if present, and expanding it similarly. This is repeated until no more core instances are present. Any instance left unclustered is deemed an *outlier*.

k -medoids: k -medoids [29] is a clustering approach that identifies *medoids*, instances considered to be most central within a cluster. The k medoids are chosen in such a way that each of the k clusters surrounding its medoid have a minimized sum of pairwise distances for each two instances in the cluster.

Instead of exploring all possible medoid-configurations, we use a (stochastic) Partitioning Around Clustering (PAM) method, varying the initial cluster medoids for each run.

Spectral clustering: Spectral clustering [34] is a clustering method that creates clusters based on similarities between instances. It transforms the instance space into a lower-dimensional space based on the eigenvalues of a similarity matrix and uses a standard clustering approach (in this case k -means clustering) on the transformed data.

In lieu of a similarity matrix, we create one using the distance matrix by convolving it with the Gaussian kernel. Similarity matrix S follows from distance matrix D as follows:

$$S_{ij} = \exp\left(-\frac{D_{ij}^2}{2\delta^2}\right), \quad (7)$$

where $\delta = 70$ represents a smoothing parameter experimen-

TABLE III: Unsupervised learning clustering matrices.

Cluster:	Standard flow descriptor				Burst-augmented flow d.			
	1	2	3	4	1	2	3	4
DBSCAN								
Win-Ffox	95	0	0	32	95	0	0	32
Win-Chr	0	127	0	0	0	127	0	0
Lin-Ffox	0	0	128	0	0	0	128	0
Lin-Chr	0	128	0	0	0	128	0	0
k-medoids								
Win-Ffox	126	1	1	0	127	0	1	0
Win-Chr	0	120	2	6	0	123	0	5
Lin-Ffox	0	0	128	0	0	0	128	0
Lin-Chr	0	0	0	128	0	0	0	128
Spectral clustering								
Win-Ffox	96	0	0	32	96	0	0	32
Win-Chr	0	128	0	0	0	128	0	0
Lin-Ffox	0	0	128	0	0	0	128	0
Lin-Chr	0	125	3	0	0	126	2	0
Hierarchical Ward								
Win-Ffox	128	0	0	0	128	0	0	0
Win-Chr	0	128	0	0	0	128	0	0
Lin-Ffox	0	0	128	0	0	0	128	0
Lin-Chr	0	0	0	128	0	0	0	128

tally determined so that the matrix is neither close to uniformly constant, nor too close to the identity matrix.

The lower-dimensional space is generated by the eigenvector corresponding to the smallest positive eigenvalue. Finally, we obtain our spectral clustering by applying a k -means algorithm on the eigenvector, where each entry corresponds to an instance.

Hierarchical Ward variance minimization clustering: Finally, we consider a hierarchical clustering approach [20], where we separate clusters based on Ward variance minimization [37]. Hierarchical clustering creates clusters from bottom to top, using the following approach. Let s and t be two clusters with $|s|$ and $|t|$ instances, respectively, jointly forming a new cluster u . Then, the distance $d(u, v)$ between u and a yet unused cluster v is a function of the distance between their hierarchical subclusters and is defined as:

$$d(u, v) = \sqrt{\frac{|v| + |s|}{T} d(v, s)^2 + \frac{|v| + |t|}{T} d(v, t)^2 - \frac{|v|}{T} d(s, t)^2}, \quad (8)$$

where $T := |u| + |v|$ is the total number of elements.

B. Performance of clustering approaches

The results in term of clustering matrices for a search of four clusters are shown in Table III. The results show the best results for each algorithm, with the best hyper-parameters for each algorithm obtained through grid search, and the DLD costs tuned individually for each method. As can be seen, there is a marked difference in performance between the different methods. DBSCAN is the only method which allows for outliers, and from the results it can be observed that only two instances were considered outliers. However, DBSCAN and spectral clustering does not provide particularly clean clusters, with cluster 2 being a mix of classes.

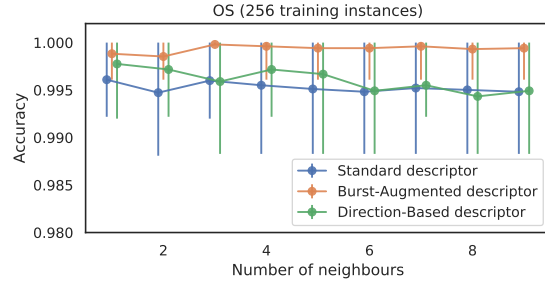
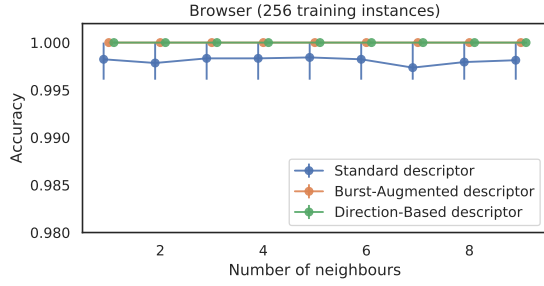


Fig. 5: Accuracy over 1- to 9-NN classification for OS and browser classification, error bars indicate the 5th and 95th percentile.

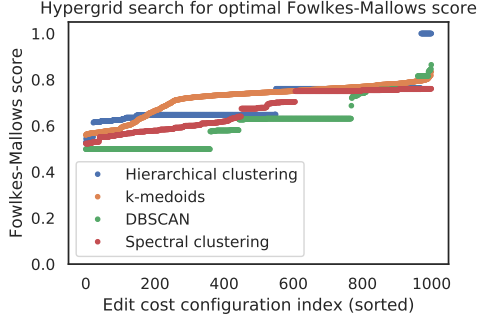


Fig. 6: Clustering performance for per-method ranked order of edit cost parameters.

In contrast, k -medoids create fairly consistent clusters, with the burst-augmented descriptor performing marginally better than the standard descriptor. The hierarchical Ward method performs the best, and creates perfect clusters using either flow descriptor.

C. Edit cost optimization

To evaluate how sensitive the clustering methods are to the configuration of edit costs, a grid search was performed over the edit cost parameter space $[c_P, c_S, c_T]$ with a resolution of $[0.1, 0.2, \dots, 1.0]$. Figure 6 shows the Fowlkes-Mallows score [14] for the 1000 edit cost combinations, sorted for each clustering method. From Table III, it is clear that the hierarchical Ward method achieves the best performance. However, it can be noted from Figure 6 that k -medoids have a higher score over a large fraction of the parameter space, indicating that there is a considerable region where it is somewhat less sensitive to edit cost parameterization than hierarchical Ward clustering.

VI. SUPERVISED LEARNING

The nature of a variable length descriptor precludes supervised learning methods such as Random Forest, SVM and others that expect a fixed dimension feature space for learning. Here, instance-based learning in form of k -NN classification is used instead.

A. The FDO k -nearest neighbor baseline results

The k -Nearest Neighbors (k -NN) classification [18] is one of the most straightforward machine learning algorithms. Given an instance x , the k training instances within minimal distance to x are located, and the predicted class is decided by the majority class of these k instances. A k -NN classifier using Damerau-Levenshtein as distance metric has been implemented. When k is even, ties in class membership among the k instances are settled by removing the instance with largest distance from the majority vote.

To establish a set of baseline results, the FDO approach is evaluated for OS and browser classification using a randomly selected stratified training set of size 256. As k -NN is used, it is of interest to study the performance over a range of values for k . Results obtained by 40 repetitions are shown in Figure 5 with the error bars indicating the 5th and 95th percentile of accuracy over the 40 repetitions. The results include three different flow descriptors: the standard and burst descriptors examined in the clustering evaluation, and additionally the direction based descriptor. The results show that FDO achieves excellent classification performance over the range of 1-9 neighbors for both OS and browser classification. There seems to be no consistent trend for the performance over the varying k . Although all descriptors provide very strong accuracy performance, the burst-augmented descriptor appears to have a slight edge.

The above OS and browser evaluation covered two binary classification problems. To examine how FDO performs with a multi-class problem, we create four composite classes consisting of the OS and browser combinations. Still using 256 training instances, the decreased number of training instances per class, and the increased number of possible classes of the neighbors in k -NN, is expected to make the classification problem harder. This can, however, not be seen from the confusion matrix in Table IV. The overall accuracy with four classes is 99.7% which is similar to what can be observed in Figure 5 for the direction-based descriptor when averaged over both OS and browser performance.

B. Training size dependence

The results presented so far were obtained with 256 training instances. These instances cover a range of variation with regards to link shaping, specific video watched, and browser

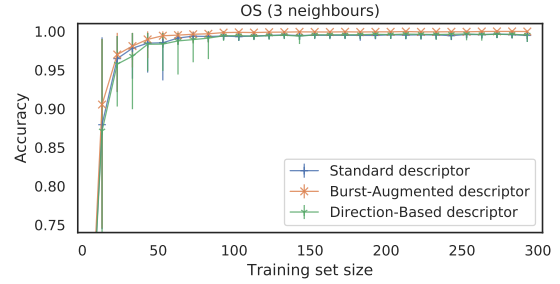
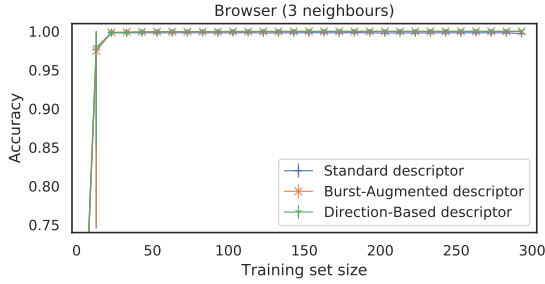


Fig. 7: Training size dependence for OS and browser classification, error bars indicate the 5th and 95th percentile.

TABLE IV: Four-class confusion matrix for 3-NN, $W^{\text{direction}}$.

		Predicted			
		Win-Ffox	Win-Chr	Lin-Ffox	Lin-Chr
Actual	Win-Ffox	1.000	0.000	0.000	0.000
	Win-Chr	0.000	1.000	0.000	0.000
	Lin-Ffox	0.014	0.000	0.986	0.000
	Lin-Chr	0.000	0.000	0.000	1.000

window size. When there is a binary classification task for OS or browser, these instances can be seen as providing 128 variations of behavior within each respective class. When considering the construction of a classification model the amount of training instances needed to obtain the desired target accuracy is often an important consideration, as the collection of training data with ground truth may be expensive or time-consuming.

An evaluation was performed to examine FDO classification performance over a range of training set sizes. The total number of training instances were varied between 8 and 300, with the remainder of the instances used for testing. This procedure was repeated 40 times to obtain both averages and measures of variation. The results are shown in Figure 7 and it is clear from the graphs that a low number of training instances is enough for excellent performance; for the browser classification one requires only around 30 instances, while the OS classifications requires around 80. There are no strong differences between the different flow descriptors, albeit that the burst-augmented flow descriptor seems to perform slightly better in classifying the OS.

C. Temporal stability results

To explore to what extent the classification results are stable between data collected at different times, a cross data set evaluation was performed. Various updates to the operating system code, browser applications, or video provider back ends may over time create differences in the flow characteristics used for classification. Here, such effects were investigated by the use of a second data set, collected with the same approach but three months later. A cross data set evaluation may serve as an indication to what extent changes occurring over that time influences the classification performance. The results of training on the first data set and classifying on the second are shown in Figure 8 as inter-datasets results. For

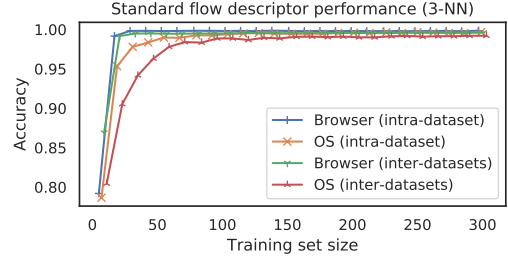


Fig. 8: Cross data-set classification performance.

comparison, the results from classifying on the first data set are also included. The graph shows slightly worse results, but especially for larger training set sizes the performance hit is relatively small. Thus, the FDO method appears to be robust with respect to time, at least over the moderate time spans tested here. To handle changes in the data over time various concept drift approaches [25] can be employed. For k-NN classifiers in particular, [24] proposes a self adjusting memory approach.

D. Robustness of classifier

In order to examine the robustness of the supervised classifier, we analysed the sensitivity of the classification process to different types of noise in the session descriptors. Three types of noise have been considered; (1) random insertion and deletion of certain flows descriptors, to measure robustness with respect to different background processes, (2) randomly substituting certain flows descriptors for other ones, to measure robustness with respect to structural changes to the streaming process and (3) randomly swapping the order of certain flow descriptors, to measure robustness with respect to temporal variations. Each of these noise representations has a corresponding operation in the DLD computations and affects the distance matrix in a different way. In Figure 9, the performance of the classifier is illustrated for different levels of noise applied to the test data. The training data was left untouched. The noise ratio represents the ratio of flow descriptor characters changed in the session descriptor. To mitigate the risk that the number of inserts cancel out the number of deletes, both values are drawn from a Poisson distribution. It can be observed that even for high values of

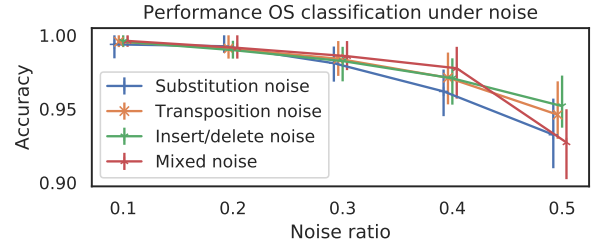
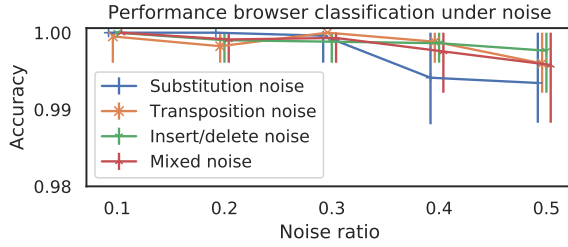


Fig. 9: Effects of different types of noise on classification performance, error bars indicate the 5th and 95th percentile.

noise, the classifiers perform relatively well. It can be noted that adding noise to a test instance tends to increase its distance to the NN training instances belonging the correct class, but to some degree also to instances of the incorrect class.

VII. DISCUSSION

Other work which considers the session level characteristics of groups of flows is the traffic causality graphs (TCG) proposed by Asai et al. [8]. As one among several case studies, they examine four different browsers and show that TCG can exploit the temporal causality arising from relevant discriminating aspects, such as DNS prefetching, to differentiate between the different browsers. However, they confirm their findings with manual inspection and do not consider how the TCG representation should be integrated with unsupervised or supervised learning for browser discrimination. Later work [7] on TCG uses graph mining to construct feature vectors which is used with an 1-NN classifier, but the evaluation was performed to discriminate between seven different P2P applications rather than browsers.

This work can be extended in several directions. The current FDO approach searches for the optimal edit operation costs for each operation. This resulted in a considerable performance improvement over the base case of equal costs. The same approach can be applied at a finer granularity, where a cost vector is learnt for the insertion, deletion and transposition costs for each letter in the flow descriptor alphabet, and a cost matrix for the substitution costs. As a full search of such parameter space is prohibitively computationally expensive, an appropriate meta-heuristic approach would be necessary, such as genetic algorithms [17] or particle swarm optimization [21].

Additionally, more elaborate discretization can be employed. In this work the relatively straightforward equal frequency discretization approach was used. However, supervised discretization methods, such as the recently proposed KSD approach [16], have been shown to perform better from a feature discretization perspective and could thus further improve FDO in relation to the setup examined here.

A related approach could be pursued by expressing the problem in terms of inter-neighbour distances. That is, a distance vector D_n is computed towards n reference instances. In essence this transforms the session descriptors from variably sized vectors, to be positions in an n -dimensional space. With such a transformation into a fixed dimensionality it becomes

possible to apply a range of classification methods (random forest, SVM, ANN) that require fixed dimensionality. It would also be possible to still maintain the use of k -NN classification, but now employ metric learning techniques to further improve classification performance. One such metric learning approach is proposed by Weinberger et al. [38], which strive to learn an optimized Mahalanobis distance.

VIII. CONCLUSIONS

This work proposes the novel Flow-Discretize Order (FDO) approach for describing session characteristics in an intuitive manner, while also retaining flow ordering information. By utilizing string distance metrics, such as the Damerau-Levenshtein, on the FDO session descriptors it is possible to perform both unsupervised and supervised learning. Here, we utilize FDO as a tool for OS and browser identification coupled to a particular user activity, in this case watching YouTube videos. The FDO approach allows for flexible construction of flow property descriptors, by using different flow properties and applying appropriate discretization. The individual flow descriptors are concatenated to form session descriptors, and by computing a distance matrix various unsupervised learning methods can be applied. Our experiments show that Hierarchical Ward clustering performs best at learning the session structure created by variation in OS and browser behavior during a YouTube viewing session.

From a supervised learning perspective, our experiments show that these high-level session characteristics, which can be derived from NetFlow records, allow a k -NN classifier to achieve very high accuracy on the classification tasks of OS and browser identification. An advantage of the scheme is that it needs only a small amount of training examples.

In summary the presented FDO approach constitutes a new way of performing high-level traffic characteristic inference. The FDO framework provides multiple promising avenues for further research and improvements such as improved methods for discretization boundary placement, more elaborate feature selection approaches, and using more fine-grained Damerau-Levenshtein costs.

ACKNOWLEDGMENTS

The authors wish to thank Sandvine for assisting with data collection. Funding for this study was provided by the HITS project grant from the Swedish Knowledge Foundation.

REFERENCES

- [1] Z. Afzal, J. Garcia, S. Lindskog, and A. Brunstrom, "Slice distance: an insert-only levenshtein distance with a focus on security applications," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, 2018, pp. 1–5.
- [2] A. Aksoy and M. H. Gunes, "Operating system classification performance of TCP/IP protocol headers," in *2016 IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops)*. IEEE, 2016.
- [3] A. Aksoy, S. Louis, and M. H. Gunes, "Operating system fingerprinting via automated network traffic analysis," in *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2017, pp. 2502–2509.
- [4] H. F. Alan and J. Kaur, "Client diversity factor in HTTPS webpage fingerprinting," in *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '19. ACM, 2019.
- [5] B. Anderson and D. McGrew, "OS fingerprinting: New techniques and a study of information gain and obfuscation," in *2017 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2017, pp. 1–9.
- [6] O. Arkin, "A remote active OS fingerprinting tool using ICMP," *login: the Magazine of USENIX and Sage*, vol. 27, no. 2, pp. 14–19, 2002.
- [7] H. Asai, K. Fukuda, P. Abry, P. Borgnat, and H. Esaki, "Network application profiling with traffic causality graphs," *International Journal of Network Management*, vol. 24, no. 4, pp. 289–303, 2014.
- [8] H. Asai, K. Fukuda, and H. Esaki, "Traffic causality graphs: profiling network applications through temporal and spatial causality of flows," in *2011 23rd International Teletraffic Congress (ITC)*. IEEE, 2011.
- [9] P. Auffret, "SinFP, unification of active and passive operating system fingerprinting," *Journal in computer virology*, vol. 6, no. 3, pp. 197–205, 2010.
- [10] B. Claise, "Cisco Systems Netflow Services export version 9," 2004.
- [11] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Communications of the ACM*, vol. 7, no. 3, 1964.
- [12] P. Eckersley, "How unique is your web browser?" in *International Symposium on Privacy Enhancing Technologies*. Springer, 2010.
- [13] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.
- [14] E. B. Fowlkes and C. L. Mallows, "A method for comparing two hierarchical clusterings," *Journal of the American statistical association*, vol. 78, no. 383, pp. 553–569, 1983.
- [15] F. Gagnon, B. Esfandiari, and L. Bertossi, "A hybrid approach to operating system discovery using answer set programming," in *10th IFIP/IEEE Int Symposium on Integrated Network Management*, 2007.
- [16] J. Garcia and T. Korhonen, "Efficient distribution-derived features for high-speed encrypted flow classification," in *Proceedings of 2018 ACM SIGCOMM Workshop on Network Meets AI & ML*, 2018, pp. 21–27.
- [17] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.
- [18] J. Goldberger, G. E. Hinton, S. T. Roweis, and R. R. Salakhutdinov, "Neighbourhood components analysis," in *Advances in neural information processing systems*, 2005, pp. 513–520.
- [19] A. Gómez-Boix, P. Laperdrix, and B. Baudry, "Hiding in the crowd: an analysis of the effectiveness of browser fingerprinting at large scale," in *Proceedings of World Wide Web Conference*, 2018, pp. 309–318.
- [20] T. Hastie, R. Tibshirani, and J. Friedman, "Hierarchical clustering," *The elements of statistical learning*, vol. 2, 2009.
- [21] J. Kennedy, "Swarm intelligence," in *Handbook of nature-inspired and innovative computing*. Springer, 2006, pp. 187–219.
- [22] D. Lay, S. Lay, and J. McDonald, *Linear Algebra and its Applications*. Pearson, 2012.
- [23] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet Physics Doklady*, vol. 10, no. 8, 1966, pp. 707–710.
- [24] V. Losing, B. Hammer, and H. Wersing, "KNN classifier with self adjusting memory for heterogeneous concept drift," in *2016 IEEE 16th international conference on data mining (ICDM)*. IEEE, 2016, pp. 291–300.
- [25] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.
- [26] P. Matoušek, O. Ryšavý, M. Grégr, and M. Vymřátil, "Towards identification of operating systems from the internet traffic: Ipfix monitoring with fingerprinting and clustering," in *2014 5th International Conference on Data Communication Networking (DCNET)*. IEEE, 2014, pp. 1–7.
- [27] J. Muehlstein, Y. Zion, M. Bahumi, I. Kirshenboim, R. Dubin, A. Dvir, and O. Pele, "Analyzing HTTPS encrypted traffic to identify user's operating system, browser and application," in *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 2017.
- [28] G. Navarro, "A guided tour to approximate string matching," *ACM computing surveys (CSUR)*, vol. 33, no. 1, pp. 31–88, 2001.
- [29] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert systems with applications*, vol. 36, no. 2, pp. 3336–3341, 2009.
- [30] S. V. Petrović and J. D. Golić, "String editing under a combination of constraints," *Information sciences*, vol. 74, no. 1-2, pp. 151–163, 1993.
- [31] S. Ramírez-Gallego, S. García, H. Mourriño-Talín, D. Martínez-Rego, V. Bolón-Canedo, A. Alonso-Betanzos, J. M. Benítez, and F. Herrera, "Data discretization: taxonomy and big data challenge," *Data Mining and Knowledge Discovery*, vol. 6, no. 1, pp. 5–21, 2016.
- [32] Z. Shamsi, D. B. Cline, and D. Loguinov, "Faulds: A non-parametric iterative classifier for internet-wide OS fingerprinting," in *Proc. of ACM SIGSAC Conference on Computer and Communications Security*, 2017.
- [33] Z. Shamsi, A. Nandwani, D. Leonard, D. Loguinov, Z. Shamsi, A. Nandwani, D. Leonard, and D. Loguinov, "Hershel: Single-packet OS fingerprinting," *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, 2016.
- [34] X. Y. Stella and J. Shi, "Multiclass spectral clustering," in *Proceedings of the 9th IEEE Conference on Computer Vision - Volume 2*. IEEE, 2003, p. 313.
- [35] B. Trammell and E. Boschi, "An introduction to ip flow information export (ipfix)," *IEEE Communications Magazine*, vol. 49, no. 4, pp. 89–95, 2011.
- [36] M. Wand and M. Jones, "Kernel smoothing," *Monographs on Statistics and Applied Probability*, London, 1995.
- [37] J. H. Ward Jr, "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Assoc.*, vol. 58, no. 301, 1963.
- [38] K. Q. Weinberger, J. Blitzer, and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," in *Advances in neural information processing systems*, 2006, pp. 1473–1480.
- [39] T.-F. Yen, X. Huang, F. Monrose, and M. K. Reiter, "Browser fingerprinting from coarse traffic summaries: Techniques and implications," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2009, pp. 157–175.