k-NN Classification of Malware in HTTPS Traffic Using the Metric Space Approach

Jakub Lokoč[1], Jan Kohout[2], Přemysl Čech[1(✉)], Tomáš Skopal[1], and Tomáš Pevný[2]

[1] SIRET Research Group, Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague, Prague, Czech Republic
{okoc,cech,skopal}@ksi.mff.cuni.cz
[2] Department of Computer Science and Engineering, FEE, Czech Technical University in Prague, Cisco Systems, Inc., Cognitive Research Center in Prague, Prague, Czech Republic
{jkohout,tpevny}@cisco.com

Abstract. In this paper, we present detection of malware in HTTPS traffic using k-NN classification. We focus on the metric space approach for approximate k-NN searches over dataset of sparse high-dimensional descriptors of network traffic. We show the classification based on approximate k-NN search using metric index exhibits false positive rate reduced by an order of magnitude when compared to the state of the art method, while keeping the classification fast enough.

Keywords: Similarity search · k-NN classification · Intrusion detection

1 Introduction

Network Intrusion Detection Systems (NIDS) are presently an essential tool in detecting intrusions in computer networks, infected computers within, and other types of unwanted behaviour (e.g. exfiltration of company's sensitive data, using peer to peer networks, etc.). Traditionally, these systems have relied on the *signature matching* paradigm, which identifies known sequences of bytes (signatures) in packets unique for a particular virus, trojan, or other threat, which we further refer to as a *malware*. The advantage of signature matching is very low false alarm rate, however, malware nowadays implements plethora of evasion techniques such as polymorphism and encryption to randomize byte sequences rendering signature matching ineffective.

While randomizing byte sequences is relatively simple, randomizing behaviour is conceptually significantly more difficult problem. For example if the attacker wants to steal data from a computer, he needs to transfer them over the network. Similarly if he wants to know, if infected computers are still infected, they need to contact attacker's computer. Behaviour based NIDS detects such actions that are specific to malware activity, by using higher-level features of the traffic, such as the number of connections to different hosts during some

M. Chau et al. (Eds.): PAISI 2016, LNCS 9650, pp. 131–145, 2016.
DOI: 10.1007/978-3-319-31863-9_10

period of time, number of transferred bytes between two computers, etc. The drawback of behaviour based NIDS is higher false alarm rate, but the recall can be higher due to robustness with respect to simple randomization of byte sequences. The other advantage of relying on higher-level features is that they are exported by most network devices (switches, routers, HTTP proxies), which increases the visibility into the network (there are more collection points), and simplifies deployment as no adaptation of devices is needed.

This paper focuses on detecting secure HTTP (HTTPS) connections related to a malware activity, which is a pressing problem due to the generally growing volume of HTTPS traffic on the Internet (accelerated by Snowden's affair) and increasing adoption of HTTPS protocol by malware for its primary mean of communication. Information about HTTPS connections are effectively limited to the number of uploaded and downloaded bytes and a duration of the connection, which makes the classification of HTTPS connections particularly difficult. Nevertheless the prior art [13,14] has presented a statistical *fingerprint* of servers based on modelling joint distribution of properties of all connections to it. It has been demonstrated that they are usable for detecting malware and grouping servers with similar purpose.

In this paper we built upon these fingerprints, as features they rely on can be extracted from HTTPS connections. We show that (i) albeit the large dimension of fingerprints (14641), the problem of separating malware connections from legitimate is not linearly separable; (ii) a simple nearest neighbour based detector have order of magnitude better false alarm rate than the linear detector at the same recall; (iii) since we use large number of labeled data, the presented results estimate well the accuracy the representation of fingerprints can offer; (iv) we demonstrate that modern indexing structures allow to implement otherwise costly nearest-neighbour based detector efficiently, such that it become competitive to the linear classifier, particularly if its better false alarm rate is taken into the account.

2 Challenges in Detection of Malware Using HTTPS

The web proxy logs have become a widely used source of input data for the network intrusion detection systems because they provide relatively lightweight information about network that can be processed in high volumes to detect suspicious communication. However, the increasing usage of encrypted web communication via the HTTPS protocol hardens such detection or any traffic analysis at all. The creators of malware are aware of this fact which leads them to design the malware to use the encrypted communication as well. A commonly used way how to deal with HTTPS on web proxies is to intercept the HTTPS traffic on the proxy, decrypt it, log the information needed and encrypt it again. While the advantage is that the same detection techniques as in the case of HTTP traffic can be used, we see the main disadvantages in the computational burden on the proxy and the security concerns associated with the re-encryption of the private traffic. An alternative way is to develop new detection mechanisms that do not

rely on features available only after the decryption. This is a challenging problem because the behavior of the network connections has to be reconstructed from very limited information. If the connection requests are not decrypted by the proxy, they are usually logged just as the so called *connect* requests [11], for which the only data available are the amounts of bytes transferred in both directions and the length of the time interval for which the connection was open. This rules out, for example, any detection methods that are based on features extracted from the visited URLs. On the other hand, if a detection method capable of working with such limited information is available, the design of the web proxy can be significantly simplified, its security improved and the privacy of the users is fully preserved. Therefore, developing such a method is a highly desirable task which motived our research presented in this paper.

3 Related Work

To the best of our knowledge, the prior art on using machine learning algorithm to classify HTTPS connections and detect malware is very limited. However there is some prior art using higher-level features as is the goal here. For example, Wright et al. [22] use a k-NN classifier with sizes and directions of TCP packets carrying encrypted traffic to identify application layer protocols. Contrary to our goal, their aim is not to distinguish between benign and malware's traffic, but to identify application protocols carried by the encrypted packets. Works of Crotti et al. [9] and Dusi et al. [10] use empirical estimates of probability distributions of packets' sizes and inter-arrival times in TCP flows for identification of application protocols (e.g. POP3, SMTP, HTTP). Despite that these do not aim directly to processing encrypted traffic, their representations could be applicable to it. The limiting factor of these approaches is that they need to know the order of each packet from the start of the TCP flow. Moreover, network sensors do not usually export information or statistics about individual packets within the connection. Many works can be found in the field of behavioral detection of malware that use non-encrypted HTTP traffic. Although they do not apply content inspection, they still employ features that can not be extracted from HTTPS traffic, for example lengths of URLs or types of HTTP methods (e.g., GET or POST) — for examples of such works, see [19,20] or [16].

In the work of Kohout and Pevny [13], servers are represented by a joint histogram of tuples $(r_{\text{up}}, r_{\text{down}}, r_{\text{td}}, r_{\text{ti}})$ with 11^4 bins, where r_{up} is the number of bytes sent from the client to the server, r_{down} is the number of bytes received by the client from the server, r_{td} is the duration of the connection (in milliseconds), and r_{ti} is the time in seconds elapsed between start of the current and previous request of the same client. Each tuple describes one connection to the server, and the dynamic range of values is decreased by taking $log(1 + x)$ before creating the joint the histogram. The biggest advantage of this representation is that the information is typically exported by most network devices supporting IPFIX [8] and Netflow [6] formats, web proxies, and they are available for encrypted (HTTPS) connections. With respect to this, these fingerprints are used in our work as basic features. Their dimension is $11^4 = 14641$ for each server.

In the introduction, it is mentioned that the biggest advantage of signature based techniques is low false alarm rate. This property is crucial for practical deployment, since the problem is unbalanced with the prevalence of legitimate traffic. For example, in our experimental dataset there are 160–220 thousands of benign servers and only 600–1800 examples of those that were related to malware activity (depending on a particular testing set). Thus high false alarm rate floods the network operator with meaningless alarms, which renders the system useless. Due to this imbalance in costs of error, the performance of each classifier is measured by the false alarm rate at 50 % recall on malware [21] (further called FP-50). Formally, the value of the FP-50 error e is defined as follows:

$$e = \frac{1}{|\mathcal{I}^-|} \sum_{i \in \mathcal{I}^-} \mathrm{I}\left[f(x_i) > \mathrm{median}\{f(x_j) | j \in \mathcal{I}^+\}\right], \tag{1}$$

where $\mathcal{I}^-$ are indexes of the negative (benign) training samples, $\mathcal{I}^+$ are indexes of the expositive training samples (malware), $f(x)$ is the output of the classifier (classification score) for the sample x and I is an indicator function. The rationale behind the measure is that we are willing to miss 50 % of malware for the benefit of having extremely low false positive rate. Moreover, thanks to the using of the 2-quantile of the false negatives (which reflects the demanded 50 % recall), the median in (1) can by replaced with mean for purposes of the FP-50 minimisation. This makes the optimization computationally tractable. In [21], the exponential Chebyshev Minimizer (ECM) was presented as a suitable linear classifier optimizing the FP-50 measure. This classifier is used also in the experimental section as a baseline classification technique. Denoting $x \in \mathbb{S} \subset \mathbb{R}^d$ the training set of samples, ECM solves the following optimization problem

$$\arg\min_{w \in \mathbb{R}^d} \frac{\lambda}{2}\|w\|^2 + \frac{1}{|\mathcal{I}^-|} \sum_{i \in \mathcal{I}^-} \log\left(1 + \exp(w^{\mathrm{T}}(x_i - \mu^+)\right), \tag{2}$$

where μ^+ is the mean of the positive samples: $\mu^+ = \frac{1}{|\mathcal{I}^+|}\sum_{i \in \mathcal{I}^+} x_i$. λ is a regularization parameter that needs to be set in prior of the training. Under reasonable assumptions on symmetry of the distribution of positive samples, the mean well approximates the median and ECM optimizes FP-50 for the class of linear functions. The optimization problem (2) is solved using L-BFGS method.

This paper investigates also a simple nearest-neighbour based classifier considering the Euclidean norm L_2 as a distance between two samples. The classification rule for a test sample x is based on its k nearest neighbours from a given training set $\mathbb{S}$. The k-NN query is defined for $k \in \mathbb{N}^+$, $x \in \mathbb{R}^d$ and $\mathbb{S} \subset \mathbb{R}^d$ as:

$$kNN(x) = \{\mathbb{X} \subset \mathbb{S}; |\mathbb{X}| = k \wedge \forall y \in \mathbb{X}, z \in \mathbb{S} - \mathbb{X} : L_2(x, y) \leq L_2(x, z)\}.$$

The k-NN classifier used in this paper assigns two values to each test sample x. The first value v_1 is the number of malicious objects in $kNN(x)$ and the second value v_2 is the sum of distances to the malicious objects in $kNN(x)$. The test samples are then sorted in the multi-column manner, where the samples are first sorted by v_1 in descending order and then by v_2 in the ascending order. This sorting is the input for the FP-50 measure.

Although being currently out of fashion the advantage of k-NN classifier is its universal consistency, which means that it converges to the optimum classifier in a Bayesian sense [2]. This property enables estimation of the accuracy that can be achieved using the representation of servers proposed in [13], provided sufficient number of well labelled samples is available (more on this in Sect. 5). As the nearest neighbour classifier can be extremely slow, the next section is devoted to presentation of indexing structures, which significantly improve the classification time. Since the data considered in this work are high-dimensional vectors, an approximate k-NN search strategy has to be employed by indexes to provide practical query processing times. The approximate search, however, affects classification, because the set of nearest neighbors just approximates the real nearest neighbors. Nevertheless, in the experiments we show that the approximation of the real nearest neighbors has negligible impact on the accuracy.

4 Efficient k-NN Search Using Metric Indexing

To perform the kNN classification fast, a database index is needed that provides exact and/or approximate kNN search over large set of high-dimensional sparse descriptors of the traffic. During last decades, there have been investigated many approaches to search efficiently in huge collections of (sparse) high-dimensional vectors. The approaches use various techniques to reduce the negative effect of high dimensionality of the data. For example, the dimension reduction techniques that try to find new low-dimensional representations of the original vectors preserving distances, or, the locality sensitive hashing [12] that tries to map close vectors to the same buckets with high probability. Within the vast portfolio of database techniques implementing the two principles, in this paper we focus on the metric index (M-Index [17]) that represents a metric variant of locality sensitive hashing [18]. The M-Index is suitable for the mentioned task as it represents efficient yet extensible solution for fast similarity searches.

4.1 Metric Indexing in a Nutshell

The fundamental trick of metric indexes lies in using lower bounds that can be used to filter out irrelevant object from the search cheaply (i.e., without the need of actual distance computations regarded as computationally expensive).

A *metric space* $(\mathbb{U}, \delta)$ consists of a descriptor domain $\mathbb{U}$ and a distance function δ which has to satisfy the metric postulates of *identity*, *non-negativity*, *symmetry*, and *triangle inequality*, defined $\forall x, y, z \in \mathbb{U}$ as:

$$\begin{aligned} \delta(x,y) = 0 \quad &\Leftrightarrow \quad x = y && \text{identity} \\ \delta(x,y) \geq 0 & && \text{non-negativity} \\ \delta(x,y) = \delta(y,x) & && \text{symmetry} \\ \delta(x,y) + \delta(y,z) \geq \delta(x,z) & && \text{triangle inequality} \end{aligned}$$

In this way, metric spaces allow domain experts to model their notion of content-based similarity by an appropriate descriptor representation and distance function serving as similarity measure[1]. At the same time, this approach allows to design index structures, so-called *metric indexes* [1,3–5,17,23] for efficient query processing of content-based similarity queries in a database $\mathbb{S} \subset \mathbb{U}$. These methods rely on the distance function δ only, i.e., they do not necessarily know the structure of the descriptor representation of the objects.

Metric indexes organize database objects (descriptors) $o_i \in \mathbb{S}$ by grouping them based on their distances, with the aim of minimizing not only traditional database cost like I/O but also the number of costly distance function evaluations. For this purpose, nearly all metric indexes apply some form of filtering based on cheaply computed lower bounds. These bounds are constructed based on the fact that exact pivot–object distances are pre-computed, where a pivot is either a static or a dynamic reference object selected from the database.

We illustrate this fundamental principle in Fig. 1a where we depict the query object $q \in \mathbb{U}$, some pivot object $p \in \mathbb{S}$, and a database object $o \in \mathbb{S}$ in some metric space. Given a range query (q, r), we wish to estimate the distance $\delta(q, o)$ by making use of $\delta(q, p)$ and $\delta(o, p)$, with the latter already stored in the metric index. Because of the triangle inequality, we can safely filter object o without needing to compute the (costly) distance $\delta(q, o)$ if the triangular lower bound $\delta_{\mathrm{T}}(q, o) = |\delta(q, p) - \delta(o, p)|$ is greater than the query radius r.

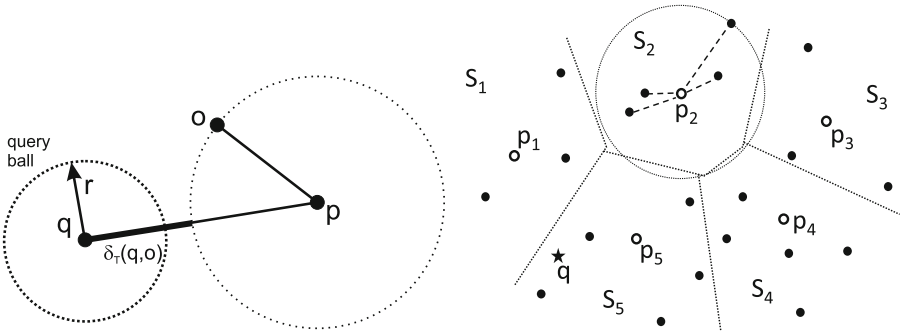

Fig. 1. (a) The lowerbounding principle, (b) The index.

4.2 Metric Index for kNN Search in High-Dimensional Sparse Data

There have been developed many metric indexes varying in the application purpose, however, in this paper we introduce a simple main-memory variant of the

[1] In our case, the descriptors are high-dimensional sparse vectors representing network traffic and the distance function is the Euclidean distance.

state-of-the-art structure M-Index that fits the requirements of network traffic kNN classification (high-dimensional data, cheap distance, many queries). Inspired by GNAT [1], M-index [17] and M-tree [5], we assume a set of p pivots p_i selected from the dataset $\mathbb{S}$, each representing a partition S_i in the metric space. The rest of the objects in the dataset are partitioned such that each object is assigned to a partition of its closest pivot. In such a way we obtain Voronoi partitioning (complete and disjoint), see Fig. 1b. Moreover, for each partition we store the distance r_i between the pivot and the furthest objects within the partition (partition radius), i.e., $r_i = \delta(p_i, o), o \in \mathbb{S}_i, \forall o_j \in \mathbb{S}_i : \delta(p_i, o_j) \leq \delta(p_i, o)$, see the dotted circle in the figure for r_2. Moreover, for each partition we also store the distances from the partition objects to the pivot p_i, i.e., $\langle \delta(p_i, o_j) \rangle_{o_j \in \mathbb{S}_i}$, see the dashed lines in the figure for $\delta(p_2, o_j)$.

The structure corresponds to one level GNAT or M-index, i.e., without additional (repetitive) Voronoi partitioning. The structure stores radius of each cluster and distances from partition objects to corresponding partition pivots as utilized by the M-Tree [5] (so called *distances to parent*). Note that also the M-Index considers the distances from partition objects to partition pivots to construct a key value that is stored in the B-Tree structure and used later for efficient searching. Since the distance function is not expensive, the structure stores information just for the most efficient filtering rules (ball-region filtering, parent filtering [23]), skipping less efficient filtering rules presented for M-Index (or GNAT) and expensive distance measures. In the following section, the approximate kNN search principles are detailed given the presented structure.

Approximate kNN Search. Given a kNN query (q, k), the index is used to process the query in a simple way. The partition with closest pivot to q is searched, then the partition with the second closest pivot is searched, and so on. During the search the closest k objects from the partitions searched so far to q are maintained as the kNN candidates. In order to speedup the search, there are several optimizations used.

First, when a partition is to be searched, its radius is checked whether the partition ball (p_i, r_i) overlaps the query ball or not. The query is formed by a ball centered in q having the radius to the actual kth nearest neighbour candidate. If the query ball and the partition ball do not overlap, the partition is skipped.

Second, whenever an object o from a partition is to be checked whether it is contained within the actual query ball or not, prior to computing the distance $\delta(q, o)$ the triangular lower bound is computed (as depicted in Fig. 1a). If the lower bound is greater than the actual query radius, the object cannot become a kNN candidate so the object is discarded without computing $\delta(q, o)$.

Third, as will be shown in experiments, it is not necessary to perform exact kNN search that could lead to exhaustive search in many of the partitions. In order to speedup the search even more, the kNN search could be stopped after a predefined number of objects is inspected (e.g., 4 % of all objects in the dataset).

For the kNN pseudocode see Algorithm 1.

Algorithm 1. kNN(q, k, S, maxObjectsInspected)

```
1: compute distance to pivots d(q, p[i]) and sort partitions S[i]
2: initialize kNN candidate set Can using pivots p[i]
3: get actual query radius r
4: count = 0;
5: // for each partition check its objects
6: foreach S[i] in S
7:     if count > maxObjectsInspected then break
8:     // query-partition overlap check
9:     if d(q, p[i]) > r[i] + r then continue
10:    foreach o[j] in S[i]
11:        // lower bound filter
12:        if |d(q, p[i]) − d(o[j], p[i])| > r then continue
13:        distance = d(q, o[j])
14:        if distance ≥ r then continue
15:        update Can by o[j]
16:        r = d(q, Can[k])
17:        count++
18: return Can
```

5 Experiments

Below experiments reflect the problem to solve, which is the identification of servers contacted by malware by observing HTTPS connections to them. The first part focuses on description of the dataset and its labeling, which is an important aspect in the network intrusion detection research. Then it moves to the comparison of linear classifier to the nearest-neighbour based one, and finishes with a discussion of its scalability.

5.1 Dataset

The experimental dataset contains logs of HTTPS connections observed during the period of one day (24 h) in November 2014 from 500 major international companies[2] collected using Cisco's cloud web security solution [7]. Each log of HTTPS connection contains source and destination IP address, number of sent and received bytes, duration of the connection, and timestamp indicating when the connection has started. Besides this, some logs contain SHA hash of the process that has initiated the HTTPS connection. These logs with hashes are used in the experiments here, since matching them with a database of malware hashes[3] provides the precious labels (malware/benign). We emphasize here that the process hash is usually missing in logs, since network devices like routers, switches, and proxies do not have any means to compute them. Also, the availability of the hash is independent to the type of the connection, therefore errors measured on this set is a good estimate of what can be expected in real world.

[2] The exact cannot be published due to non-disclosure agreements.

[3] Specifically, the hash was considered to be malicious if the corresponding process was detected by at least 20 anti-viruses used by `virustotal.com` service.

In total, there was 145 822 799 connections to 475 605 unique servers with the total volume of transferred data 10 082 GB. As already mentioned above, a request is deemed to be related to a malware activity if hash of its parent process was in Virustotal's[4] database of malware hashes. Similarly a domain or destination IP address is considered to be malware if there was at least one malware connection to it.

Since the subject of the interest are servers, a server's feature vector (a descriptor or fingerprint) is formed from a joint histogram of four-tuple $r = (r_{up}, r_{down}, r_{td}, r_{ti})$[5] of all HTTPS connections to it, as described in Sect. 3.

The employed dataset has specific properties. The 4-dimensional vectors used to create the joint histograms form a strongly uneven distribution in the corresponding 4-dimensional space, making some parts of the space (and thus corresponding bins of the fingerprints) dominant. Based on these observations, we have also calculated joint histograms considering inverse document frequency (idf) weighting. However, according to our experiments (see Fig. 2) the idf weighting just deteriorates the performance of the k-NN classifier.

5.2 Test Settings

The main error measure used for the comparison is the FP-50, which is the probability of false alarm at 50 % recall. Note that FP-50 is evaluated after all query objects with assigned labels are sorted using ranking obtained from the employed classifier. The rationale behind the FP-50 measure is that low false alarm rate is extremely important, hence the measure focuses on it. Moreover, since malware usually uses more than one server, 50 % recall is perfectly reasonable.

FP-50 and other quantities were estimated using six-fold cross-validation, where each fold contained all HTTPS connections observed during continuous four hours. Domain descriptors were calculated separately from connections in the training folds and in the testing fold. The exact number of domains (samples) is shown in Table 1.

The linear ECM classifier [21] has only one parameter, which is the regularization constant λ (see Eq. 2). Although the parameter has an influence on the accuracy, for a small values its effect on it is limited, which is caused by the large number of training samples. In all experiments below its value was set to 10^{-8}. The basic nearest neighbor classifier has also one tunable parameter, which is the number of nearest neighbors. The level of approximation of the k-NN search is another parameter affecting precision. The effect of both parameters on FP-50 measure is studied below in Fig. 4. The metric index uses 1000 randomly selected pivots.

[4] `virustotal.com`.

[5] r_{up} is the number of bytes sent from the client to the server, r_{down} is the number of bytes received by the client from the server, r_{td} is the duration of the connection (in milliseconds), and r_{ti} is the time in seconds elapsed between start of the current and previous request of the same client.

Table 1. The number of benign/malware/total servers (samples) in each fold in the cross-validation.

	fold					
#domains	1	2	3	4	5	6
in training set	444540	441357	441073	424555	428635	440690
in testing set	161908	165132	161352	224004	222294	192691
benign	160911	164276	160741	222605	220549	191204
malware	997	856	611	1399	1745	1487

5.3 Performance of the Classifiers

FP-50 together with training and classification times is shown in Table 2. The results reveal that the linear classifier (ECM) is outperformed in FP-50 by almost order of magnitude by the 4-nearest neighbor classifier (without idf weighting), which means that the problem is not linearly separable despite the high dimension. The presented times[6] show that efficient indexing techniques are necessary for k-NN classifier to reach practical times.

Table 2. FP-50 estimated by the cross-validation of compared classifiers, together with their average training and classification times.

		time of	
classifier	FP-50	training	classification
ECM	13.23 %	56mins	0.3s
exact 4-NN no index	2.015 %	0s	63mins
exact 4-NN	2.015 %	44s	17mins
exact 4-NN with idf	2.247 %	44s	23mins
approx. 4-NN (4 % DB)	2.017 %	44s	3mins

Figure 2 shows FP-50 measure of exact search based k-NN classifier with respect to k and two types of fingerprints — with and without tf-idf weighting. The advantage of a small k in the k-NN classifier can be explained by a non-linearity of the problem and very low number of malware domains, which amounts only of 0.6 % of the total number (see Table 1). Indeed, a t-sne plot [15] in Fig. 3 reveals that malware domains do not form a tight clusters, but they are scattered in the space and surrounded by legitimate ones. This means that increasing k increases the proportion of benign domains in the neighbourhood yielding into incorrect classification.

[6] The experiments have run on 64-bit Windows Server 2008 R2 Standard with Intel Xeon CPU X5660, 2.8 GHz, 12 cores supporting hyper-threading. The training of the ECM classifier has run on a virtual machine (VMWare) using 8 cores CPU 2.2 GHz and 132 GB RAM. Matlab library MinFunc has been used.

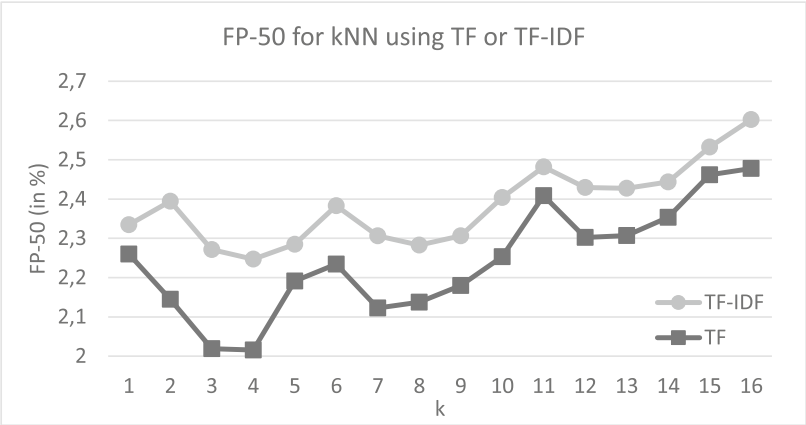

Fig. 2. FP-50 measure for exact kNN classifier and different types of descriptors.

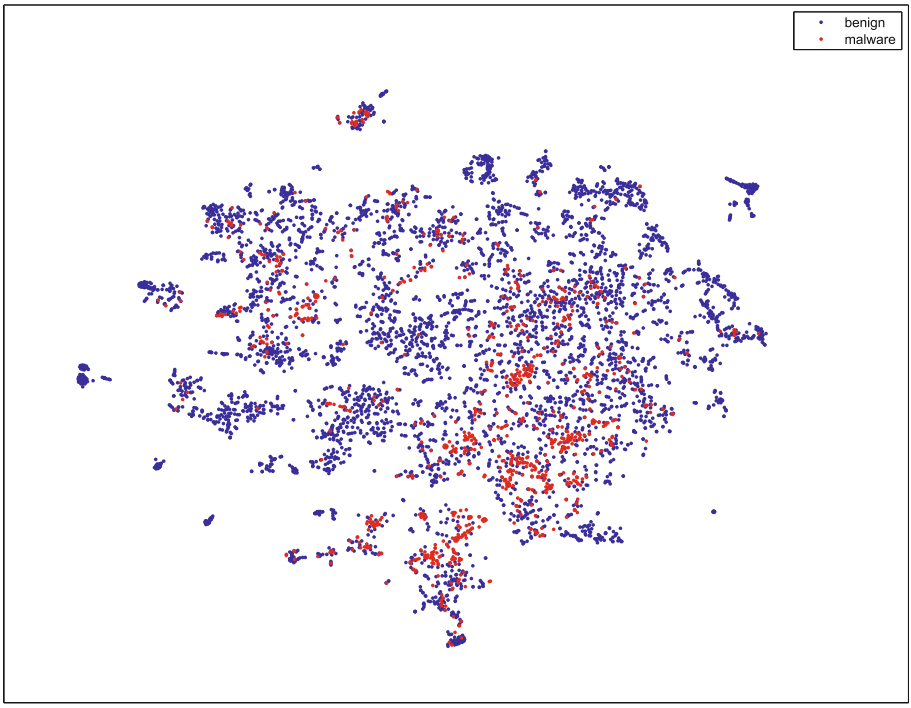

Fig. 3. A t-sne visualization of the space of servers' fingerprints, blue/red dots represent benign/malware fingerprints. The plot was created from uniformly sampled 5000 instances of benign fingerprints and 1000 instances of fingerprints of servers contacted by malware from the fold 6 of the cross-validation. This figure demonstrates that the malware's behavior does not follow a simple pattern common to all servers. However, as many of the servers contacted by malware form small clusters, the k-NN classifier which leverages the local similarities has a good chance to succeed (Color figure online).

5.4 Speeding-Up Nearest-Neighbor Search

As presented in Table 2, the classification based on exact 4-NN search, even when using the index, takes several orders of magnitude more time to perform the classification than using a liner classifier. The classification of 192.000 servers with a training set comprising 440.000 servers takes 17 min even on a 12-core server supporting hyper-threading. Table 3 presents times for exact and approximate search (only for fold 6). The exact 4-NN search using the index takes 17 min because it has to evaluate about one fifth of the distance computations. Note that the filtering power is limited because of the high-dimensionality of the data (14641). Therefore, only the approximate search strategies visiting just limited number of objects can further improve the efficiency of the retrieval using the index. For example, visiting just 1 % of the training dataset during query processing takes 17 times less time then exact search. However, the approximate search also affects the performance of the classification. Figure 4 presents FP-50 measure for the k-NN classification variants considering faster approximate search strategies for k-NN query processing. We may observe that the FP-50 measure is almost the same for exact search and for approximate search strategy visiting just 4 % of the training dataset. For the all considered levels of approximation, the optimal value of k was the same (k = 4) as for the exact search strategy.

Table 3. 4-NN classification times (in seconds) of the testing data for different levels of approximation.

% of DB	1	2	3	4	5	6	7	8	9	10	100
time	73,1	118,2	157,1	193,4	226,9	258,0	287,2	314,7	341,0	367,3	1021,6

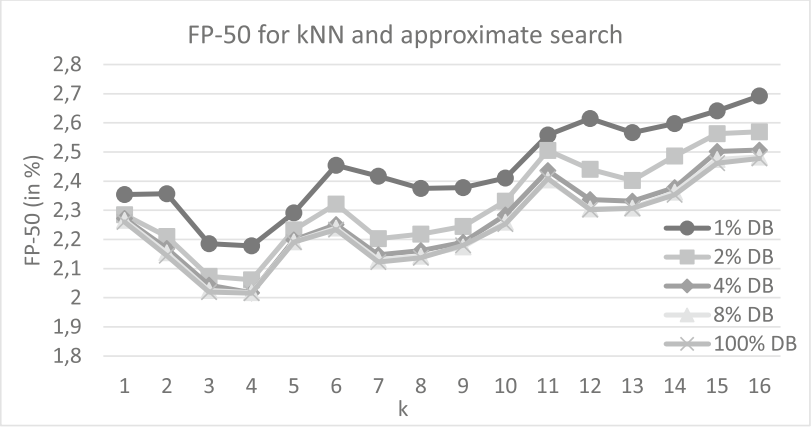

Fig. 4. FP-50 measure for the k-NN classifier and different levels of approximation.

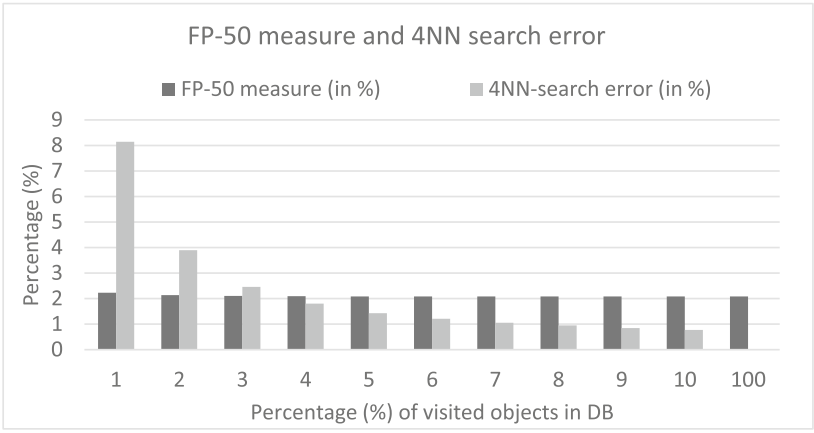

Fig. 5. FP-50 measure and approximation error for 4-NN classifier and different levels of approximation.

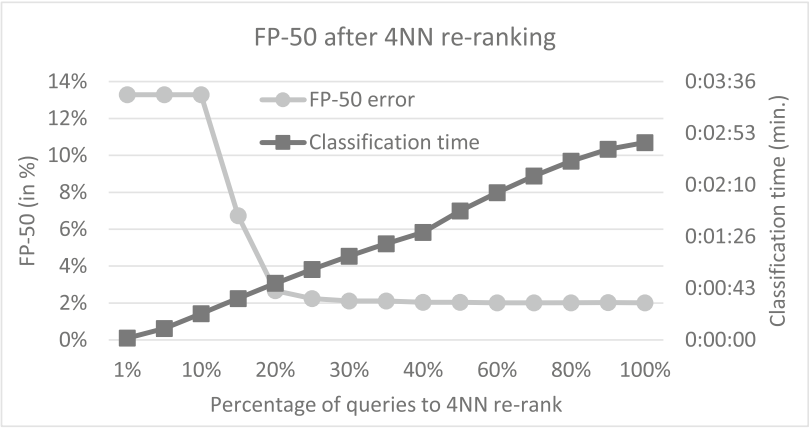

Fig. 6. FP-50 measure after re-ranking of ECM-based classification using k-NN classifier. Note that the time for classification corresponds to the secondary axis.

Figure 5 illustrates the approximation error[7] with decreasing number of visited objects during 4-NN query processing. Albeit the approximation error significantly increases for lower percentage of visited objects, the value of the FP-50 measure changes only slightly. Such advantageous trade-off is promising for our future research focusing on even faster approximate search algorithms for k-NN classification of HTTPS traffic data.

[7] For a given query, the approximation error is computed as a normed overlap distance between the query result returned by approximate k-NN search and the correct result returned by exact k-NN search.

In the last set of experiments, the linear classifier ECM was employed as an efficient ranking approach for the whole query set, while the investigated k-NN classifier was used to re-rank only a prefix of the ECM-based ordering of query objects. This technique is based on the assumption that the highly efficient ECM classifier could identify a subset of the query objects with low precision but high recall (i.e., with more than 50 % objects representing malicious servers) thus avoiding k-NN search for the whole query set. Based on the results presented in Fig. 6, re-ranking of just 30 % of the query objects using expensive k-NN classifier has almost the same effectiveness as evaluating k-NN classifier for all query objects, while the FP-50 evaluation time is almost three times lower.

6 Conclusions

In this paper, we have presented a technique for detection of malware in HTTPS traffic using k-NN classification. We have presented the efficiency of metric indexing for approximate k-NN search over dataset of sparse high-dimensional descriptors of network traffic. In the experiments, we have demonstrated that the classification based on approximate k-NN search using metric index exhibits false positive rate reduced by an order of magnitude when compared to the ECM linear classifier, while keeping the classification fast enough. We have also demonstrated that both classifiers can be combined in order to reach the overall classification time below one minute and FP-50 measure close to 2 %.

In the future, we would like to extend this work in several directions. We would like to investigate distance learning approaches for more effective classification using k-NN classifiers. We would also like to try data reduction techniques to improve both effectiveness and efficiency of the k-NN classification.

Acknowledgments. This research has been supported by Czech Science Foundation project (GAČR) 15-08916S.

References

1. Brin, S.: Near neighbor search in large metric spaces. In: Proceedings of 21th International Conference on Very Large Data Bases, VLDB 1995, 11–15 September 1995, Zurich, Switzerland, pp. 574–584 (1995). http://www.vldb.org/conf/1995/P574.PDF
2. Chaudhuri, K., Dasgupta, S.: Rates of convergence for nearest neighbor classification. In: Advances in Neural Information Processing Systems (2014)
3. Chávez, E., Navarro, G.: A compact space decomposition for effective metric indexing. Pattern Recogn. Lett. **26**(9), 1363–1376 (2005). http://dx.doi.org/10.1016/j.patrec.2004.11.014
4. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. **33**(3), 273–321 (2001)
5. Ciaccia, P., Patella, M., Zezula, P.: M-tree: an efficient access method for similarity search in metric spaces. In: VLDB 1997, pp. 426–435 (1997)

6. Cisco: Cisco IOS NetFlow. http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html
7. Cisco: Cloud Web Security (CWS). http://www.cisco.com/c/en/us/products/security/cloud-web-security/index.html
8. Claise, B., Trammell, B., Aitken, P.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information (2013). https://tools.ietf.org/html/rfc7011
9. Crotti, M., Dusi, M., Gringoli, F., Salgarelli, L.: Traffic classification through simple statistical fingerprinting. SIGCOMM Comput. Commun. Rev. **37**, 5–16 (2007)
10. Dusi, M., Crotti, M., Gringoli, F., Salgarelli, L.: Tunnel hunter: detecting application-layer tunnels with statistical fingerprinting. Comput. Netw. **53**, 81–97 (2009)
11. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1. https://tools.ietf.org/html/rfc2616
12. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: Proceedings of the 25th International Conference on Very Large Data Bases, VLDB 1999, pp. 518–529. Morgan Kaufmann Publishers Inc., San Francisco (1999). http://dl.acm.org/citation.cfm?id=645925.671516
13. Kohout, J., Pevny, T.: Automatic discovery of web servers hosting similar applications. In: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM) (2015)
14. Kohout, J., Pevny, T.: Unsupervised detection of malware in persistent web traffic. In: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2015)
15. van der Maaten, L., Hinton, G.E.: Visualizing high-dimensional data using t-SNE. J. Mach. Learn. Res. **9**, 2579–2605 (2008)
16. Nelms, T., Perdisci, R., Ahamad, M.: Execscent: mining for new c&c domains in live networks with adaptive control protocol templates. In: Proceedings of the 22nd USENIX Conference on Security (2013)
17. Novak, D., Batko, M., Zezula, P.: Metric index: an efficient and scalable solution for precise and approximate similarity search. Inf. Syst. **36**(4), 721–733 (2011)
18. Novak, D., Kyselak, M., Zezula, P.: On locality-sensitive indexing in generic metric spaces. In: Proceedings of the Third International Conference on SImilarity Search and APplications, SISAP 2010, pp. 59–66. ACM, New York (2010). http://doi.acm.org/10.1145/1862344.1862354
19. Perdisci, R., Ariu, D., Giacinto, G.: Scalable fine-grained behavioral clustering of HTTP-based malware. Comput. Netw. **57**, 487–500 (2013)
20. Perdisci, R., Lee, W., Feamster, N.: Behavioral clustering of HTTP-based malware and signature generation using malicious network traces. In: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (2010)
21. Pevny, T., Ker, A.D.: Towards dependable steganalysis. In: IS&T/SPIE Electronic Imaging (2015)
22. Wright, C., Monrose, F., Masson, G.M.: On inferring application protocol behaviors in encrypted network traffic. J. Mach. Learn. Res. **7**, 2745–2769 (2006)
23. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. Springer, New York (2005)