Czech Technical University in Prague

Faculty of Electrical Engineering

Doctoral Thesis

August 2019 ***Jan Kohout***

Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science

Representation of Communication in Computer Networks Security

Doctoral Thesis

Jan Kohout

Prague, August 2019

Ph.D. Program: Electrical Engineering and Information Technology
Branch of study: Information Science and Computer Engineering

Supervisor: doc. Ing. Tomáš Pevný, Ph.D.
Supervisor-Specialist: Martin Rehák, Ph.D., Ingénieur ECP

Acknowledgments

First, I would like to express my deep and sincere gratitude to my supervisor Tomáš Pevný and supervisor-specialist Martin Rehák who guided me in my research and were ready to discuss all the research topics with me.

My special thanks belong to Petr Somol who provided me the opportunity to finish this work during my engagement at Cisco Systems and also for giving his valuable feedback on this thesis.

I would like to thank my former and recent colleagues for their support and inspiring discussions, especially Martin Grill, Martin Kopp, Ivan Nikolaev and Tomáš Komárek.

My thanks also go to Jakub Lokoč and Přemysl Čech from the Faculty of Mathematics and Physics, Charles University in Prague, for the great collaboration during our work on the topics from similarity search in large databases.

I gratefully acknowledge the funding sources thanks to which all the scientific work presented in this thesis could be done, specifically the Grant Agency of the Czech Republic, projects number P103/12/P514 and 15-08916S.

Abstract

In current solutions for network security analysis, monitoring and management, machine learning algorithms play an important role. The increasing volume of traffic and number of devices connected to networks, rapidly changing variants of malware and also dynamically changing configurations of networks are all factors that increase the difficulty of manual analysis of network traffic. Involvement of algorithms that help to automate such processing is therefore a logical step in development of modern systems. In order to apply the machine learning algorithms effectively, proper representation of the network traffic needs to be used which can be consumed and understood by the algorithms.

While there have been many works published that employ machine learning for network traffic analysis, there have been almost no works that would study the representations of traffic in general. In this thesis, we propose a generalizing view on building network traffic representations that is independent of the any specific application scenario. We base our approach on a probabilistic look on the problem and derive several ways of representing the traffic that stem from representations and comparison of probability distributions. We put emphasis on usability of the proposed representations for different cases in network security and management which we demonstrate in experimental evaluations on data from real large networks.

We start with studying histogram representations and their applications in network outlier detection, unsupervised analysis of servers' behavior and large scale similarity search utilized for identification of network hosts infected with malware. This approach proves to be able to provide sparse representations of the traffic that can be effectively stored and processed and do dot require any optimizations of the representations' parameters beforehand.

Next, we move our attention to kernel embedding of probability distributions from which we derive an alternative representation framework. This approach enables us to represent the traffic patterns without explicit estimation of probability densities of the underlying distributions while preserving enough information. Thanks to this, better accuracy in analytical tasks can be achieved compared to the histogram representations at the cost of higher but still acceptable computational requirements. Moreover, multivariate distributions of more variables can be considered in these representations. Again, we demonstrate general usability of this framework in multiple scenarios including both supervised and unsupervised analysis of the traffic.

Finally, we present a way how to capture network entities' long-term behavior using discrete categorical features for which the previous frameworks can not be directly used. Using this representation, we experimentally show that basic characteristics of the network entities can be identified using only very lightweight information extracted from the network traffic logs.

Abstrakt

V současných bezpečnostních řešeních pro analýzu, monitorování a správu počítačových sítí hrají algoritmy strojového učení důležitou roli. Zvyšující se objem provozu a počet zařízení připojených do sítí, rychle se měnící varianty malwaru a také dynamicky se měnící konfigurace sítí jsou faktory, které zvyšují obtížnost ruční analýzy sít'ového provozu. Zapojení algoritmů, které takové zpracování automatizují, je proto logickým krokem ve vývoji moderních bezpečnostních systémů. Pro efektivní použití algoritmů strojového učení je však třeba použít především vhodnou reprezentaci komunikace, ze které mohou algoritmy získat potřebné informace.

Přestože bylo publikováno mnoho prací, které využívají strojové učení pro analýzu sít'ového provozu, neexistuje téměř žádná práce, která by studovala reprezentace komunikace v síti obecně. V této práci navrhujeme zobecňující pohled na budování reprezentací provozu v síti, který je nezávislý na jakémkoli konkrétním scénáři. Náš přístup je založen na pravděpodobnostním pohledu na problém, ze kterého odvozujeme několik přístupů, jak provoz v síti reprezentovat. Důraz je kladen na použitelnost navrhovaných reprezentací v různých případech z oblasti zabezpečení a správy počítačové sítě, což dokazujeme v experimentálním vyhodnocení na datech ze skutečných sítí.

Nejprve jsou studovány reprezentace založené na histogramech a jejich aplikace v detekci anomálií, v analýze chování serverů a pro rozsáhlé podobnostní vyhledávání použitelné pro identifikaci počítačů infikovaných malwarem. Tento přístup ukazuje, že je schopen poskytovat řídké reprezentace provozu, které lze efektivně ukládat a zpracovávat, a to bez jakékoli optimalizace parametrů těchto reprezentací předem.

Dále se zaměřujeme na tzv. vnořování pravděpodobnostních distribucí pomocí jádrových funkcí, na kterém je založen další přístup k reprezentaci komunikace. Tento přístup nám umožňuje reprezentovat chování v síti bez explicitního odhadu hustoty pravděpodobnosti daných distribucí při současném zachování dostatečného množství informace. Díky tomu je dosaženo větší přesnosti při analýze provozu za cenu vyšších, ale stále akceptovatelných výpočetních nároků, navíc lze v tomto přístupu také lépe využít vícerozměrných náhodných veličin. Všeobecná použitelnost tohoto přístupu je opět doložena několika experimenty, které zahrnují strojové učení s učitelem i bez učitele.

Závěrem představujeme způsob, jak zachytit dlouhodobé chování jednotlivých zařízení v síti pomocí diskrétních příznaků, pro které nelze předchozí metody přímo použít. S využitím této reprezentace je ukázáno, že základní charakteristiky zařízení lze odhalit pouze pomocí velmi omzené informace, kterou lze získat ze záznamů provozu v počítačové síti.

Contents

List of Figures

List of Tables

1

Introduction

Securing computer networks has been without any doubts recognized as a very complex and challenging task. While the ultimate goal is clear — to identify any threats that my compromise the network and to prevent them from getting to the network — it is composed of many partial tasks that might be very diverse. Not only the knowledge of malicious software itself, but also an insight into the protected network and its functional parts is needed in order to apply the security measures effectively. Naturally, many tools have been developed that help the administrators and security specialist to get such knowledge and insight. Among other solutions, the purpose of Intrusion Detection Systems (IDS) deployed in computer networks is to detect presence of unwanted or malicious activities in the network and report them to the operator.

Traditionally, the IDS have been divided into two groups: the host-based IDS and network-based IDS. While the host-based IDS detects presence of the malicious activity on single hosts in the network, the network-based IDS gathers information about traffic in the entire network and detects the malicious activities based on this information. Both types of IDS are often deployed concurrently in corporate networks to improve security countermeasures. Traditional approaches used in the IDS for threats detection were based on static signatures matching — the activities of network hosts were recorded and then matched against a database of known signatures of malicious activities. If a matching activity was found in the database, the IDS raised an alarm. The signatures matching approaches were traditionally valued for their simplicity, results that were easy to interpret by the operators and low false positive rates of the detection. However, during the last years, the malware landscape has been changing faster and faster and also the total volume of traffic in networks has been growing. Moreover, usage of encrypted network communication to protect users' privacy has been on the rise, too. Each of these trends hardens the application of signature matching solutions — new signatures have to be published with very low delays in order to capture rapid changes of malware and also the encryption brings more obstacles as the signatures-based solutions often use content inspection to extract the signatures from the observed traffic. For these reasons, especially the network-based IDS begun to adopt detection algorithms that employ machine learning to identify malicious activities based only on high level information about the hosts' behavior and without inspecting the contents of communication. Nowadays, machine learning algorithms that can be found in an IDS include various methods from anomaly detection to clustering and classification. Regardless of the specific algorithm which is applied, a suitable representations of the users' and machines' behavior must be first created and then passed to the detection algorithm. The network traffic is usually analyzed at different levels of granularity, from single packets to complex descriptions of behavior of each user or server. At each level, some representation of the behavior of the observed entities is created and processed.

However, the detection of malware's activity or presence is not the only task of current security solutions. Simultaneously with the detection of malware presence, the administrators and network operators need to react on increasing complexity of the networks. Due to many

trends like the Internet of Things (IoT) or bring-your-own-device (BYOD), the structure of networks is becoming more unclear and very hard to manage manually. The insight of administrators into such rapidly changing networks is therefore limited which hardens detection of policy violations, identification of suspicious devices in the networks and other security events that might become a starting point of a security incident. Moreover, many services that used to be operated within the managed network like data storages or accounting systems are now being moved into third-party cloud solutions which imposes new challenges in detection of insider threats and data exfiltrations. In order to keep such environment manageable, solutions that help to monitor behavior of the network need to be developed. These methods include monitoring of users' behavior and automatic identification of anomalies in it, algorithms that automatically check users' identities based on their behavior (so called User and Entity Behavior Analytics — UEBA) to prevent user accounts misuses or solutions like Cisco Tetration [30] that are used to monitor cloud services usage. Similarly as an IDS, also these solutions designed for automatization of network management and network structure discovery leverage machine learning algorithms for which the accurate representations of behavior is crucial.

1.1 Use-cases of machine learning in traffic analysis

Similarly as in other domains like images processing and computer vision, natural language processing or text analysis in which machine learning methods were widely adopted for processing of data, also in the domain of computer networks security we can find many different ways how machine learning contributes to improved performance of the target system and reduces the burden that is put on human operators. Below we list the main use-cases in which the machine learning is typically employed for analysis of network traffic for security reasons. Therefore, in all of these cases a good representation that allows comparison of the analyzed traffic is essential.

1.1.1 Anomaly detection

The anomaly detection (or outlier detection) gathers methods that aim at identification of those data samples that are somehow *very different* for the majority. According to [70], a statistical outlier "is an object which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism". In network security, for example, it means that behavior of a machine infected by malicious software should be statistically different form behavior of those that are not infected, given a proper representation of their behaviors is used. Many different algorithms for detection of anomalies have been proposed for various domains of application [2]. However, anomaly detection methods — when used on their own — often suffer from relatively high number of false alarms (so called false positives) [64]. Therefore, techniques for further dealing with the false positives from anomaly detection have been developed. In [63], a complex system is presented in which outputs from many different anomaly detectors are aggregated in a multi-layered architecture to reduce false positive and create a network intrusion detection system whose detection core is based on anomaly detection. Similarly, Cisco's Cognitive Threat Analytics [136] — a security product with worldwide deployment which processes billions of network traffic log entries per day — uses anomaly detection consisting of about 40 detection algorithms as its first layer filter to select the most anomalous events that are passed for further processing by the system.

1.1.2 Traffic and behavior classification

Unlike the anomaly detection, whose aim is to detect possibly unknown but outlying samples, classification — as a part of the supervised learning methods family — aims at assigning a

specific label to each classified sample. The label can be either binary (e.g., malicious or benign activity, infected or not infected machine etc.) or multi-class (e.g., family of malware to which given sample belongs or a name of the communicating application). There is a large variety of classification tasks in analysis of network data that can include, for example, identification of malicious relations between a client and a server using multiple instance learning together with support vector machines (SVMs) [49], usage of decision trees for multi-class classification of malware categories using logs from web proxies [17] or identification of application layer protocol from sequences of packets [145]. More complex surveys on this topic can be found in [101] and [140]. The above mentioned Cognitive Threat Analytics [136] system uses multiple different classification algorithms in its second layer to classify the most anomalous part of the traffic preselected by the anomaly detection.

1.1.3 Clustering and unsupervised analysis of data

Besides finding the outlying data objects or classifying them to target classes that are known in advance, finding a structure and similarities in a corpus of initially unknown data plays an important role as well. Clustering algorithms — as representatives of unsupervised learning methods — are employed to discover groups (called clusters) of objects that are mutually similar to each other while they are dissimilar to objects from other groups. No matter which specific algorithm is used to achieve this, there is a need to properly represent the clustered objects such that this representation allows to assess similarity (or distance) of two objects. This is a necessary precondition to form meaningful clusters that would then allow discovery of new patterns in the data. Clusters discovered by a clustering algorithm can be then used to define new target classes for which classifiers can be trained, can be passed to anomaly detectors where they are used to build better baseline of normal behavior (outliers are expected to be dissimilar to most fo the clusters) [85] or simplify analysis of the data by humans by focusing on the entire clusters instead of single objects. In the domain of computer networks security, clustering has been used to identify communication channels of bots [66] or to group malicious binaries for identification of their underlying malware families [128]. Various algorithms can be used, depending on the properties of objects' representations and definition of the similarity or distance function. If the representations live in a metric space, then the k-means algorithm [87] is a popular choice, if only pair-wise similarities are available, then spectral clustering [100] can be applied or even algorithms for identification of communities in graphs can be implemented [16]. For example, the latter one was used in [81] to discover clusters of domains that are contacted by similar malware samples.

1.2 Thesis goals and key contributions

The previous section provided a brief overview of use-cases in which machine learning can be applied in analysis of network traffic. As we suggested, for all the use-cases — no matter which specific algorithm is used — it is essential to be able to correctly represent the network communication such that it captures the important patterns of the traffic or the modeled entities and this information can be utilized by the algorithms. While some solutions might use representations that are tightly connected with the specific algorithm used on top of them, it is in general very practical to design the representations independently of the specific machine learning algorithm. For example, there are many algorithms for clustering of data and most of them operate only with the notion of similarity of dissimilarity between pairs of objects.

Hence, given a representation that allows for evaluation of pair-wise similarities or dissimilarities between objects, different out-of-the-box clustering algorithms can be immediately applied, evaluated and compared to select the one which is the most suitable for the given scenario. Correspondingly, an anomaly detection algorithm operates with the notion of similarity, too — it looks for objects, the outliers, that are dissimilar to the others. And finally,

a classifier can be seen as looking for objects that are similar enough to the pattern characteristic for the target class such that it can be classified as belonging to that class.

Building on these observations, the main goal of this thesis is to propose methods for representation of communication in computer networks that are easily portable and adjustable for different use-cases while allowing the algorithms to evaluate similarity or dissimilarity between two communication samples. Specifically:

- We introduce an underlying model of network communication from which the representations can be derived in different scenarios and use-cases. The model treats the communication as identically and independently distributed messages drawn from a probability distribution which characterizes the given communication.
- We propose two main representation frameworks based on that model and present their applications — a histogram-based representation which excels in its sparsity and scalability and a representation derived from kernel embedding of probability distributions which significantly improves performance of algorithms working on top of it.
- We experimentally show on data from real computer networks that the representations are able to provide solid background for algorithms that leverage pair-wise similarities between the analyzed objects and improve the results achieved by prior methods.
- In case of each proposed representation, we show that they fulfill the following main requirements: The are general enough, such that they allow to model and compare samples of communication from different sources of data, at various levels of abstraction and are independent of any specific algorithm working on top of them. This is important, because representations having these properties allow effective development of modular analytical systems, enable easy comparisons of individual algorithms and can be quickly deployed when new types of input data or modeled entities appear. Therefore, we experimentally test the representations on different types of network traffic logs and together with algorithms for anomaly detection, clustering as well as classification.

1.3 Outline of this thesis

With respect to the goals presented above, this thesis is structured as follows:

- In **Chapter 2**, we review the TCP/IP stack of network communication protocols and representations of network traffic that can be found in prior art works which employ those representations as input for different machine learning algorithms. These algorithms have been deployed in the prior art works for analysis of traffic or detection of malware's presence.
- In **Chapter 3**, we introduce a formal definition of so called *message sets* — a universal term introduced here to refer to any snapshot of communication that is about to be represented and analyzed. The formal definition is derived form a probabilistic view on the network communication and the representations discussed in ongoing chapters are all designed with respect to this unifying view. The model views the traffic as composed of identically and independently distributed messages. Using this assumption which does not incorporate order of the messages, we are able to model traffic even in case when the order of messages is random (e.g., servers contacted by multiple users).
- In **Chapter 4**, we describe all the datasets that are used for experimental evaluations throughout this work.
- **Chapter 5** then describes an approach for representation the traffic, which is based on joint sparse histograms. We present multiple scenarios in which the histogram representations are used for anomaly detection or unsupervised analysis of the network traffic logs. We specifically study an extension of the definition of standard histograms which decreases the effect of strict quantization and we show that this type of histograms outperforms the standard ones in the experimental evaluation.

- In **Chapter 6**, we use the histogram representation on large scale data. We design a method for effective similarity search on large data, demonstrating that the histograms can be used together with optimized similarity search on large databases of network traffic representations. Moreover, this chapter presents usage of the histogram approach specifically for malware detection in encrypted HTTPS (Hypertext Transfer Protocol Secure) which is considered very challenging.
- **Chapter 7** then describes representations that build upon kernel embedding of probability distributions. The experimental results presented in the chapter show that these representations can help to achieve significantly better performance of an algorithm working on top of the representations when compared to the histogram representations at the cost of increased storing requirements. However, these requirements are still much lower in the prior art representations that were derived from the kernel embedding of distributions, too.
- **Chapter 8** extends the aim of this work to cover also discrete domains by presenting an approach to representation of persistent behavior of network entities using a dictionary of categorical features.
- Finally, **Chapter 9** concludes this work, summarizes the key achievements and shows possible ways of further research based on the results presented in this thesis.

2

Domain background and representations used in prior art

In this chapter, we first review the well-known and established stack of protocols from the TCP/IP family. In vast majority of current networks, including the Internet, all the communication is implemented and organized with respect to this layered protocol stack. Therefore, reviewing it here helps to establish all the terms used in further chapters. Moreover, it helps to put the reviewed representations into context as they typically work at one or more levels of that stack and we often refer to these layers.

After reviewing the TCP/IP stack, we move to overview of network traffic representations used in prior art works. The idea of modeling behavior of network communication has been identified as useful for different applications and studied in many prior art works. Regarding the works that aim at detection of malware's presence, we emphasize the publications that aim at detection of so called command & control (C&C) channels of malware — the communication channels used by malware creators to coordinate the activity of malware and to possibly build and remotely maintain whole networks of computers infected with malware that participate in coordinated illegal activities — so called botnets. Due to the mechanisms used to manage the botnets, the representations of long-term communication channels play very important role in the process of their detection. We review works from this area separately in Subsection 2.2.

The second group of related works in which representation of the communication plays very important role include works whose goal is to identify an application layer protocol (such as HTTP, SMTP or FTP) or even the concrete communicating application from the traces of network communication. These solutions are important, for example, for automatic policy violation detection and for identification of unwanted applications in a network. We review these works in Subsection 2.3.

2.1 TCP/IP stack

The TCP/IP family of protocols — the TCP/IP stack or TCP/IP model — is a layered suite of network communication protocols that define communication of peers in a computer network [133]. The model solves processing of the communication from the level of local transmission of the data in a local network up to the level of end-to-end communication between individual user applications that need to interchange data. As already mentioned, the whole TCP/IP stack is composed of multiple layers, where each layer specifies communication at one specific level of abstraction. Specifically, the model is composed of the four following layers [18]:

- **Link layer** — The lowest layer of the model solves communication within one local network. This is called the *link* in the TCP/IP terminology, hence the name of the layer. The link layer takes data from the upper layers and encapsulates them to so called *frames* that are then passed to the hardware implementation of the local network to be

transmitted on the media. A frame is a basic unit of communication at this layer. To identify who is the transmitter and who is the receiver of the given frame, link layer addresses are used — for example, the MAC (media access control) addresses in 6-byte format like:

$$00 : 0d : 83 : b1 : c0 : 8e$$

where the first three bytes identify the manufacturer of the given network adapter and the last three bytes identify the specific instance of the adapter. Besides that, special MAC addresses exist, e.g. broadcast addresses used to broadcast frames to all peers on the link.

- **Network layer** — The network layer (or the IP layer) handles communication between hosts that are in different local networks. According to the principle used in the entire design of the TCP/IP stack, it accepts data from the upper layers and encapsulates them into IP packets that are sent between the networks. To deliver packets to the desired hosts, the IP layer uses IP addressing scheme, either the IPv4 or IPv6 [108, 36], and so called packets routing. While the purpose of IP addressing is to uniquely identify each host within the different networks (e.g., within the Internet), the purpose of packets routing is to provide rules how to forward each packet on its way through different networks such that it successfully reaches the target host. The IP addresses are hierarchically organized numerical identifiers (for example, a 4-byte identifier like 173.38.220.43 in case of IPv4 scheme) of the hosts and the routing process uses this addressing to find way for each packet thorough a system of multiple connected networks. The routing is handled by routers, which are devices at the edges of networks that receive and forward the packets from/to the neighboring networks. The rules that determine where to forward each packet based on its destination IP address are stored on the routers in a form of routing tables. The tables are maintained and updated using routing protocols (like RIP, OSPF, BGP and others) [108, 8].
  The IP layer sends each packet independently of other packets and it is up to the upper layers to verify that all data were successfully transmitted. The motivation is to avoid the burden of maintaining the packet streams when it is not necessary (if it is really needed, then it is implemented within the above layers).
- **Transport layer** — This layer extends the inter-hosts communication to service-to-service communication. Therefore, this layer distinguishes different services (applications) running on each host and conveys communication between them. To distinguish individual services on one host, the transport layer introduces so called *ports*, which are integer identifiers that specify the services within each host. The most used protocols at this layer are the Transmission Control Protocol (TCP) [109] and the User Datagram Protocol (UDP) [107]. While UDP is a very lightweight protocol which technically provides really just an extension in the from of port numbers on top the IP layer, TCP is much more complex protocol used to convey reliable end-to-end streams of packets between two hosts. As such TCP ensures that all the packets in the stream were received correctly and handles initialization and termination of each connection. Therefore, it offers a reliable tunnel through which two applications running on different hosts can communicate. Which of these protocols to select for communication depends on what kind of application is performing the communication. Fro example, downloading resources for a web page like images will likely use the TCP protocol because there will a larger amount of data transferred and both sides of the communication will need to ensure that all packets transferring individual parts of an image were transferred successfully. On the other hand, for lightweight services such additional control is not necessary. For example, in case of the Domain Name System (DNS) which is used to resolve IP address of a host given its hostname, both the query and the answer will likely fit into one packet. Therefore, no maintenance of the communication stream is needed and lightweight UDP is used for this purpose.

The port numbers that identify the services are 16-bit unsigned integers. Theoretically, an arbitrary port can be used for any service. However, to simplify orientation in the world of inter-networking, the lowest port numbers were assigned and reserved for well-known services. The list of these assignments is maintained by the Internet Assigned Numbers Authority (IANA) organization. For example, the web traffic (HTTP protocol) is expected to use port number 80, the DNS service uses the port number 53 etc.
Because the whole communication streams can be observed at this layer of the TCP/IP model, many traffic logs that are further processed by analytical systems are generated at this level. For example, the Cisco NetFlow records or IETF IPFIX [73] provide aggregated information from this layer. Similarly, many of the further reviewed works utilize logs from this layer.

- **Application layer** — This layer contains protocols that specify format of the application data exchange. Using the protocols provided by the transport layer, the application layer protocols specify format of messages interchanged between the applications, e.g. between a user's web browser and a web server. This layer contains a large variety of protocols. To provide some examples, the Hyper Text Transfer Protocol (HTTP) [47] is definitely one of the most prevalent protocols and was originally proposed for transferring data of web pages. It defines messages used by a client to request a specific resource of a web page, to provide additional information about its configuration etc. We discuss properties of this protocol in more detail in Subsection 2.1.1. Among HTTP, the DNS (Domain Name Service) protocol for resolving IP addresses for devices' hostnames, the SMTP (Simple Mail Transfer Protocol) for handling electronic mail (e-mail) or Telnet for remote terminal are other examples of widespread application protocols.
  Similarly as at the transport layer, many types of log records can be created at the application layer and then used for security analytics. The advantage is that the logs can contain application-specific fields that can provide very useful information. For example, web proxy logs are produced at this layer that provide information about HTTP connections observed in the network. Logs of this type are used also in this thesis for multiple experiments presented in further chapters.

The entire TCP/IP model is based on the principle of data encapsulation. At each layer, a packet (sometimes called a datagram) from an upper layer is treated as data part of the packet at the current layer. The current layer then appends its header to this data and passes the whole packet to the lower layer. Each layer's header contains layer-specific information about the packet that are used by that layer on the packet's way through the network (or networks). This is illustrated in Figure 2.1. Similarly, when a frame is received at the link layer, it goes through the upper layers and at each layer, the header is removed and the packet is processed with respect to the information contained in the respective header. For packets that are only forwarded to other hosts it means that not all the layers need to be actively involved — the routing is done at the network layer, hence when a device discovers in the packet's IP header that its destination is not the current host, it immediately forwards it according to the rules in its routing table without any processing at the upper layers.

2.1.1 HTTP(S) connections and traffic logging

In this subsection we review in more detail the way how web proxies produce their logs from the application layer of the TCP/IP stack. The reason why we specifically discuss this logging here is that datasets used for experimental evaluations in this thesis are mostly created based on the web proxy logs. Hence, it is important to review the basic properties of the logging process here.

The Hyper Text Transfer Protocol (HTTP) became a very popular and widespread application layer protocol for communication and transferring data over Internet (not only for web pages for which it was originally designed but also for many other applications). In its original form, it is based on plain text messages interchanged between a client and a (web)

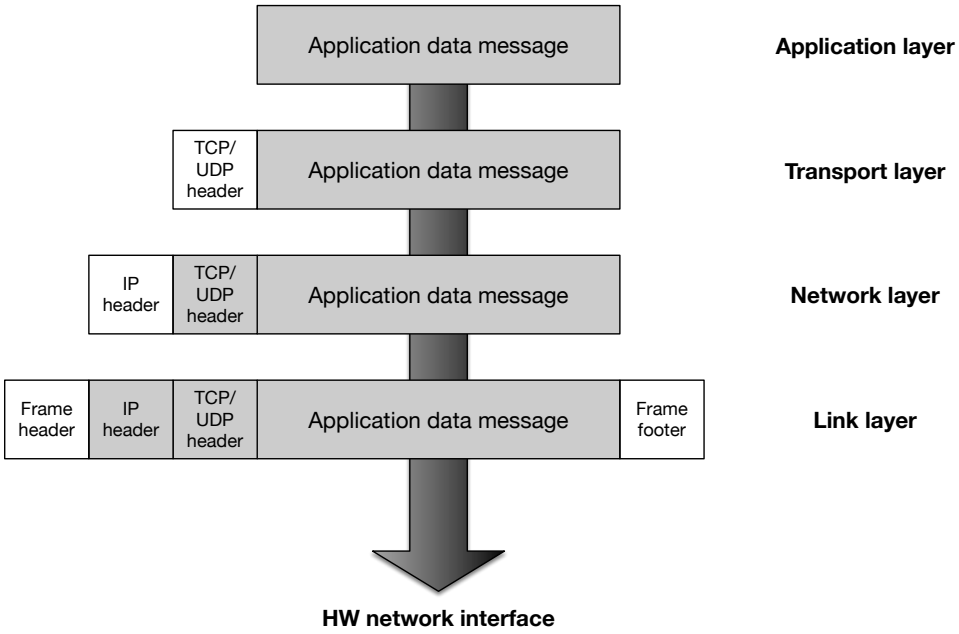


Fig. 2.1: Encapsulation of data by individual layers in the TCP/IP model. At each layer, packet from the upper layer is treated as the data part (grey color in the figure) and the layer-specific header is appended.

server. This allows easy implementations of both the client and the server side as well as logging of interesting characteristics of the connections at a web proxy.

Web proxies are devices commonly used in corporate environments that act as gateways between the clients and the web servers. Therefore, they are able to see all the HTTP headers (defined in [47]) of requests and responses and log the information extracted from them. These logs are then used as input data for various analytical systems. Besides the amounts of transferred data and timings, a web proxy can log the UserAgent string (a string identifying the software which issued the request), the exact URL which was contacted or the referrer string (an URL from which the client was redirected to the current URL). All such information can become very helpful in forensic analysis of the data and for discovering security incidents. Despite that the proxy is potentially able to see the contents of the transferred data (for example, the contents of a requested web page), such information is not directly logged due to both capacity and privacy reasons.

However, in case of encrypted HTTPS protocol, which improves users' privacy by encrypting the communication between the client and the server using additional TLS (Transport Layer Security) sub-layer (or SSL - Secure Sockets Layer which is the predecessor of TLS), the proxy — as the encryption is designed as end to end between the client and the server — is not able to log the specifics of the HTTP protocol as it only sees the transport layer data, i.e. characteristics of the TCP tunnel through which the client communicates with the server. An example of a single datum provided by a web proxy for an HTTPS tunnel is shown in Table 2.1. When a client in an environment equipped with the proxy wants to contact a web sever using HTTPS, the client typically uses a CONNECT method to ask the proxy to arrange the connection for him/her but only at the transport layer of the TCP/IP stack. This scenario is illustrated in Figure 2.2. The proxy then continues to maintain the TCP tunnel through which this communication is sent but is not able to distinguish individual requests and responses. Therefore, it might happen that multiple requests are sent in the same tunnel which are not directly observable by the proxy.

timestamp	1467816731
user	j.smith
URL	https://www.google.com/
duration	253
requestBytes	120
responseBytes	560
port	443

Table 2.1: Example of fields logged by a web proxy for one HTTPS request-response pair.

Solutions to overcome this limited insight have been proposed. These are mainly based on "man-in-the-middle"-like inspection of traffic going through the proxy. If the networking environment is set to do so, the proxy can intercept the connections by decrypting the communication, logging the necessary information and encrypting it again. This, however, requires full trust of the clients in the proxy as this concept goes against the the end-to-end encryption of the communication between a client and a target server. Additionally, it introduces new security risks because the proxy becomes a single point of failure in the sense that if the proxy is compromised by an attacker, privacy of all the users in the network is at risk no matter that they use HTTPS. Furthermore, the users have very limited capabilities to detect such compromise. Finally, it is usually not possible to analyze all the encrypted connections in this manner due to their total volume. Therefore, the proxy has to select only a limited portion of connections that will be analyzed in this way according to some rule or heuristic. Because of these limited capabilities for inspection of the HTTPS traffic by a proxy, it is very challenging to develop methods that would be able to analyze traffic logs for the encrypted connections.

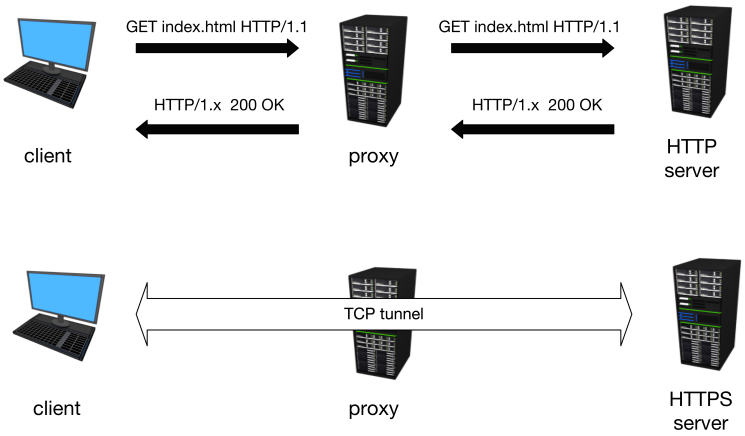


Fig. 2.2: Illustration of differences between HTTP and encrypted HTTPS communications going through a web proxy device. While in case of HTTP the proxy is able to see HTTP headers of individual requests and responses, in case of HTTPS there is only a TCP tunnel established through which the encrypted communication is sent directly between the client and the server.

2.2 Representations used for C&C channels detection

Botnets — the networks of machines (called bots) connected together to allow their collaboration and co-ordination — have been of high interest in the security research community

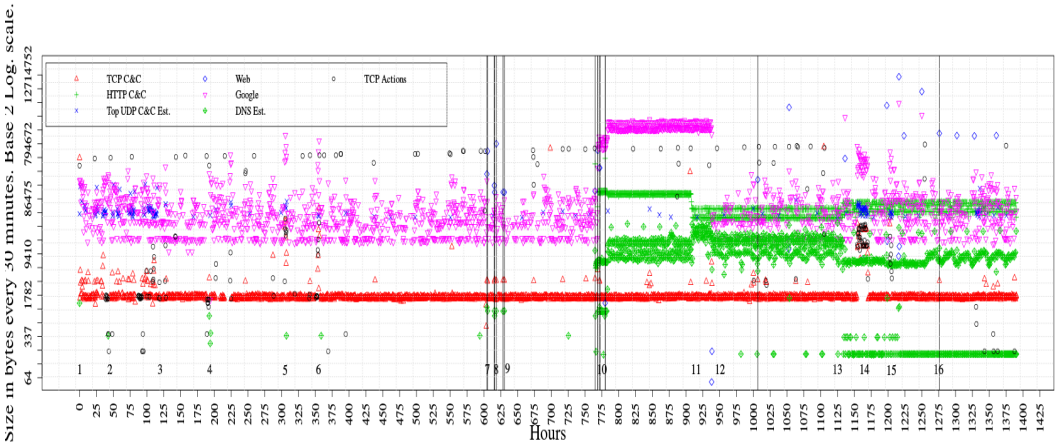


Fig. 2.3: Capture of the network activity (transferred bytes) of a bot during a period of 57 days. The communication within the C&C channel is marked by green (HTTP) and red (TCP) color. This figure demonstrated the high regularity of the particular C&C channel with respect to the transferred bytes in time. This figure was originally published in [53], we reuse it with permission from the authors of the original work.

because of their computational power and potential. When the infected machines are misused for a malicious activity like sending spam, denial of service attacks or sensitive information harvesting and exfiltration, the impact on the target of that malicious activity can be indeed devastating. Therefore, significant effort has been invested into development of techniques capable of detection of machines infected with such malware. Despite that the purposes of botnets can differ, the detection techniques mostly rely on basic properties of the botnets which is the mechanism used by the coordinator of the botnet (the botmaster) to send instructions to the bots and receive data gathered by the bots. The need to manage the bots implies that there has to be a communication channel between each bot and the botmaster through which the instructions and data are transferred. These channels are referred to as the command and control (C&C) channels. As the botmaster typically needs to keep the bots operable for a longer period of time, the C&C channels often have a long-term nature hence the communication within a C&C channel can be observed repeatedly in time. Because of the specialized purpose and the need to coordinate the bots effectively with minimal delays, the C&C channels often exhibit specific regularities in their behavior. For this reason, the methods designed for detection of botnets' communication often employ some representation of the behavior of communication channels discovered in the network to identify possible C&C communication. The importance of modeling the long term behavior of the C&C channels is emphasized in [53] in which the authors deeply studied the behavior of one concrete C&C channel for a period of 57 days. Based on their observations, the authors created a stateful model which characterizes the behaviour of the C&C channel. This shows a good chance that the behavior of a C&C channel can be well captured by a representation which models its communication patterns in time. This is also illustrated in Figure 2.3 which shows the behavior of the bot malware studied in [53]. We can see that the C&C communication (green and red) is very regular (unlike the communication to Google which is marked by purple color) with relatively low variability of the amounts of bytes transferred in one hour.

During the recent years a large volume of works dealing with the behavioral detection of C&C channels has been published. Therefore, it is not feasible to review here all the representations of communication that were proposed in those works. However, a common trait of the vast majority representations used in the prior art is the assumption that C&C will exhibit some kind of regularity which is taken into account when the features are designed.

Below, we review several selected approaches as a representative sample of the prior art. Surveys containing references to other detection methods can be found in [46, 7] and [54].

BotMiner

This detection framework was described in [66] and it is a very complex system designed for detecting C&C channels by correlating communication patterns of network hosts, working at the level of TCP/UDP flows (e.g., IPFIX or Cisco NetFlows). The entire system is composed of several stages each of which assesses different aspect of the network hosts' behavior. One of these stages is focused on discovering communication channels, called C-flows, that could be the C&C channels of the same botnet due to their mutual similarity. To identify these groups of similar C-flows it employs two-step clustering of their vector representations. The vectors are built in the following way: One C-flow is defined as a set of flows that share the same source IP address, destination IP address and destination port observed within a predefined epoch of time, e.g., one day. Therefore, it typically represents the communication channel between a user in the local network and a server identified by the tuple `[destination IP, port]`. The following four features are extracted from the flows within each C-flow:

- Number of flows per hour
- Number of packets per flow
- Average number of bytes per one packet
- Average number of bytes per second in a flow

The frequencies of observed values of each feature are captured by a histogram with 13 bins whose centers are set using 13 quantiles of the global distribution of the feature values observed in all C-flows. Finally, the numerical vector representing the given C-flow is obtained by concatenating the histograms for all the features. Therefore, each C-flow is represented by a 52-dimensional real vector.

BotFinder

The work [134] uses similar definition of the communication channel as BotMiner mentioned above. As it also works at the level of TCP/UDP flows, it defines the communication channel as a sequence of flows (chronologically ordered) between a local endpoint (e.g., an IP address in the local network) and a remote endpoint (e.g., a tuple consisting of the remote server's IP address and port). As the authors want to avoid the inspection of the communication contents, it uses only features based on the sizes and timings of the flows within a communication channel. Specifically, it uses these features to represent each communication channel:

- Average inter-arrival time between two consecutive flows
- Average duration of one flow
- Average number of bytes sent in a flow from the local endpoint to the remote endpoint
- Average number of bytes sent in a flow from the remote endpoint to the local endpoint
- FFT value — this feature reflects the empirical observation that C&C channels tend to be very regular thus there is a good chance that the communication will be periodic. The value of this feature is computed in the following way: The communication channel is divided into smaller time windows and the communication within the channel is treated as a binary signal in which the value 1 represents that there was some communication observed in the given time window and the value 0 represents that there was no communication present in the time window. Consequently, the power spectral density of the fast Fourier transformation ([56, 31]) of that signal is computed and the most significant frequency is used as the feature value.

Disclosure

The system called Disclosure described in [14] was designed for identification of C&C servers from Cisco NetFlow traces. To identify possible C&C servers it represents the behavior of each server using features extracted from NetFlows going to or from that server. Therefore, it also represents the behavior of the communication channels to these servers. The features are designed under similar assumptions as in the previous works hence they are created to capture regularities in sizes and timings of the NetFlows. The features used to represent the behavior of the server include:

- Mean and standard deviation of the servers' flows sizes, separately for the incoming and outgoing flows
- Autocorrelation ([67]) coefficients computed from the time series containing sizes of incoming and outgoing flows' of the server
- Minimum, maximum, median, and standard deviation of inter-arrival times of flows from the same client to the server
- Unmatched flow density which is defined as the difference between the number of flows incoming to the server and the number of flows outgoing from the server. The motivation for this feature is to enable detection of C&C channels that try to contact servers that are no longer reachable, which can happen relatively often due to takedown of already discovered servers

Other works

Furthermore, the work [150] represents behavior of packet streams within 5-minute time windows as numerical vectors and use them for training classifiers that detect C&C channels utilizing peer to peer (P2P) architectures of communication. The features introduced in the paper include average size of packets' payload (but deep packet inspection is not applied), variance of the packets' payload size, number of exchanged packets, number of transmitted packets per second, size of the first packet in the stream, and average inter-arrival time between packets.

Among the other remaining works from this area, we will mention here the work [45] which also uses the features derived from mean values and standard deviations of sent and received bytes or packets within a communicational channel but accompanies them with the value of so called *persistence* of the communication channel. The persistence value is computed by the algorithm proposed in [59] and it counts the number of sub-windows within a longer time window in which the communication was active. This reflects the fact that the C&C channels need to be long term thus the communication within the channels occurs repeatedly though not necessarily with exact periodicity. This way of measuring persistence motivated our work presented in Chapter 8 of this thesis. The authors used time windows composed of 24 sub-windows, and each sub-window had length of 10 minutes for this measurement.

2.3 Representations used for applications identification

In this section, we review several selected works that also employ modeling of behavior of communication in the network but they do not aim exactly at detecting communication belonging to malware. Again, the volume of works published in this area is large, therefore we review just a representative selection here which provides insight into the approaches adopted in prior art. Dominant topics studied by the works in this category are the problems of identifying the communicating application or application layer protocol (e.g., SMTP or FTP) by fingerprinting the communication without content inspection. The problem of identification of an application communicating over the network without inspecting the communication's contents became more intensively studied during the last years, because the amount of encrypted communication is increasing which notably hardens the utilization of

traditional approaches based on deep packet inspection. Moreover, the approaches relying on assumption that the applications will use ports assigned by the standards can easily fail because there is no guarantee that an unknown application will use the standard port, either intentionally to mask itself as another application or because some type of tunneling is used.

Reviews of selected prior art works are summarized in the paragraphs below.

The packets' sizes and inter-arrival times extracted from TCP flows are used to build statistical fingerprints of application layer protocols in [33]. The flows are defined as streams of packets that share the same communicating endpoints. The fingerprints are based on probability density functions (PDFs) of tuples containing the packets' sizes and inter-arrival times. The fingerprinting works in the following way: each packet in a flow is represented by a pair $(s_i, \Delta t_i)$ of values characterizing its size s_i (in bytes) and inter-arrival time Δt_i elapsed between the given packet and the previous packet in the flow. Each protocol's fingerprint is then composed of L estimations of PDFs where the i-th PDF captures joint distribution of the packets' sizes and inter-arrival times estimated from the $(s_i, \Delta t_i)$ pairs representing the i-th packets in each flow belonging to the given protocol. The motivation for using L different probability density functions to represent each protocol is, as stated by the authors of the paper, the demand to capture also order of the packets in the flows of the fingerprinted protocol. The authors refer to the proposed algorithm as TunHunter. As this is one of the methods that we used in experimental comparison further in this thesis, it is also discussed in Chapter 7.

Similar features to those used in the previously reviewed work [33] are used in [41] to represent behavior of TCP flows that are also treated as ordered packet streams between two endpoints. The goal of the work is to use this representation to train a classifier which is able to identify tunneling of one application over the application layer protocol which was not originally developed for that application. The authors provide concrete examples of tunneling different applications over SSH and HTTP protocols. The rationale behind the representation proposed in their work is that the sizes and timings of packets that transfer some payload are expected to be determined by the communicating application, especially in the early phase of the session. One flow f, composed of r packets represented by pairs $(s_i, \Delta t_i)$ similarly as in the previous work, is represented as follows:

$$f = \begin{pmatrix} s_1 & s_2 & ... & s_r \\ \log(\Delta t_1) & \log(\Delta t_2) & ... & \log(\Delta t_r) \end{pmatrix}$$

For each known application A, these representations are used for estimation of probability densities $p(f_i|A)$ which are the probability distributions of values $(s_i, \log(\Delta t_i))$ observed at the i-th positions of packet sequences (i.e., flows) belonging to the application A. The classifier then uses these probability densities to compute likelihood for an unknown flow which appears to the system as belonging to the application A (e.g, by using the transport layer port which is normally assigned to application A) to assess whether the the flow indeed belongs that application or whether it belongs to another application which tries to mask itself as A .

The work [145] has the similar goal as the work [33] — to identify application protocol, namely HTTP(S), AIM, SSH, SMTP and Telnet, from the stream of TCP packets carrying encrypted data. As the authors assume that the data are encrypted, they use only sizes of packets and their directions (i.e., from client to server or from server to client) within the

stream to represent the protocols' behavior. They propose two different representations in the work:

The first representation is based on counting packets of several different types within a time epoch to create behavioral profiles of the protocols. Based on its size and direction, each packet is assigned to one of M predefined types and the behavioral profile of the protocol in time epoch s is created by counting the packets of each type that were observed during the epoch s in a stream belonging to the given protocol. Therefore, the profile of the protocol for one time epoch s is a histogram with M bins in which the value of i-th bin represents the number of packets of type i that were observed during the epoch s. The authors use $M = 4$ in their experiments (distinguishing packets based on their direction and testing whether the size of the packet is under 64 bytes or above). Furthermore, in order to represent the behavior of the protocols in multiple time epochs, the histograms from consecutive epochs are put in a sequence which is then the final representation of the protocol's profile within a longer time period.

The second approach is based on Hidden Markov Models (HMM) [10] and is inspired by models used for protein sequences alignment in bioinformatics. The modeled packet sequences are treated as Markov chains with one hidden Markov state for each packet position in the sequence. The emitted symbols in each hidden state are the M possible packet types described in the previous paragraph. The proposed HMM uses four hidden states: `Server Match` and `Client Match` whose probability distributions for symbols emission match the structure of sequences which was learnt from the training sequences of the given protocol (either for packets traveling from server to client (`Server Match`) or for packets traveling from client to server (`Client Match`)). The remaining two hidden states are called `Insert` and `Delete` and their purpose is to model cases in which one or more packets were inserted into or deleted from the sequence of packets. The authors use these two states to model retransmissions of the same packets (`Insert`) and packets lost in the network (`Delete`).

Besides the works reviewed in more detail above, many other publications can be found in the area of network protocols and applications fingerprinting that employ some kind of representation of behavior of the communication to achieve their goals. For example, in [86], similar features as mentioned above, namely the distributions of sizes and directions of packets in encrypted streams, were used for profiling web pages. Very similar approach as in [33] was used by the same authors in [32] for detection of HTTPS tunnels. Hermann et al. in [71] aims at fingerprinting of websites from streams of TCP packets captured during loading of web pages to identify which web page was accessed by encrypted communication. The classifier is implemented as Naive Bayes with histograms of packets' sizes as the feature vector. SPID [72] is a modular framework for identification of application layer protocols (BitTorrent, HTTP, SSH,...). Depending on the specific source of data, it can use different number of features. Independently of the particular feature set which is used, observed values of each feature are modeled separately by a histogram. The classifier is implemented as a nearest prototype classifier with distance defined by Kullback-Leibler divergence between the known protocol models and a representation created from the classified sample of communication. These two last works ([71] and [72]) were also included in our experimental comparison, hence they are further discussed in Chapter 7.

The work [98] proposes a representation of TCP flows based on aggregate statistics of transferred packets and bytes in each flow (like means and variances of times and sizes or selected TCP parameters). This representation is then used by an SVM classifier to identify communicating application, browser or operating system. A disadvantage of this representation is that it is not easily extendable to cases when it is needed to represent unordered sets of packets or flows (e.g., for representing servers instead of individual flows as in our experiments in Chapter 5).

In [141], a mechanism based on Hidden Markov Models (HMM) is introduced to build profiles of users behind a NAT (a network address translation device like, for example, a gateway router of a local network or its part). The representation uses only elementary fea-

tures like sizes, packet counts and inter-arrival times of NetFlows to represent users' behavior. However, the training of the underlying HMMs is supervised (needs a training set with labels determining which NetFlow belongs to which user), which makes the representation unusable for cases with little or no labelled data.

The paper [34] deals with the problem of identifying the application layer protocol when the port-based identification is not reliable enough. Specifically, the authors focus on distinguishing HTTP(S), SMTP, FTP, BitTorrent, msn, netbios-ssn, oms and IMAP4 protocols in a scenario when all these protocols are using TCP port number 80 and no content inspection is possible. The authors propose using directions and sizes of the first four packets in a TCP flow as features. The method then uses a set of heuristic filters and decision trees to distinguish the protocols. The work [43] considers the similar scenario (identification of an application protocol from packet streams) with the same features (sizes and directions of the first n packets) but uses different classification techniques — namely SVMs, Gaussian Mixture Models (GMMs) and a decision tree as well. Furthermore, it considers the reject option for cases when an unknown protocol (a protocol which was not in the training data) is encountered.

In the work [50], the authors aim at identification of Skype traffic within the HTTP traffic, because Skype uses the communication over the port 80 as well to prevent blocking on firewalls (similarly as many other applications). The work introduces so called HTTP workload model using features observed at the level of HTTP request-response pairs, specifically: request size, response size, inter-arrival time of requests, number of requests per page and page retrieval time. Using these features, model of normal HTTP behavior is created. Given a set of new request-response pairs, statistical tests — namely the χ^2-test or Kolmogorov-Smirnov test — are then used to identify whether they come from the normal HTTP traffic or not (which might indicate the Skype traffic).

The aim of [3] is to identify operating system family and its version by fingerprinting network communication of the classified device. The authors use features from multiple layers of the TCP/IP model — the IP time-to-live (TTL) value (extracted from headers of IP packets), list of TCP options extracted from TCP packets, selected extensions extracted from Client Hello packets used by the TLS protocol (a protocol used for encryption of communication at the application layer) and User-Agent strings observed in HTTP requests.

Despite the high number of works in this area, the limited information available when no content inspection is applied cause that the features used for the communication representations are often very similar among these works. Besides the selected works reviewed above, this is also the case with works [120], [116], [12] or [83] with the exception that in the latter work, the representation is enriched with information extracted from payload data which involves the content inspection.

2.4 Summary of the prior art

There have been many works published in both of the areas of the reviewed publications. The works mostly differ in the proposed algorithms but regrading the features used for representations of the communication, the differences are not very significant. This is mainly caused by the limitation resulting from the fact that we are focused on works that do not use content inspection for modeling of the communication. As a consequence, the information about volume of the transferred data and timing of the communication are typically used. One of the reasons to do so is that this type of information is always available. Concerning the way how the features are used to build the representations, the differences are slightly more distinctive. Despite that, there are not many works (if any) that would focus primarily on the representation of the communication itself. The representations proposed so far are created based on empirical experience only and to fit the specific algorithm or application scenario. To best of our knowledge, there has been no work which would focus just on the design of

the communication representation in general, and which would try to formalize the concept of the communication and justify its representation with some theoretical background.

Based on the reviewed works, we can introduce an abstraction of the modeled communication — we can view it as an (un)ordered sequence of (possibly bidirectional) *messages* interchanged between the communicating endpoints during a certain period of time. The messages form the basic building blocks of the communication and they can be, for example, single packets, individual flows or any other basic actions, depending on the specific scenario in which the communication is modeled. A common aspect of all the representations is that the features used to build them are extracted with respect to these messages. This therefore forms the basis of the general view on the communication modeling that we present in this thesis and is formalized in the next chapter.

To conclude this chapter, we present an overview of the representations found in prior art in Table 2.2 in which we summarize the basic types of messages and features used for building behavioral representations of network communication in each work, providing references to particular works that use them.

Example works	Messages	Features
[33], [41], [32]	packets	sizes and inter-arrival times
[141]	flows	sizes, packet counts, inter-arrival times
[145], [86], [34], [43], [71]	packets	sizes and directions
[66]	flows	volume-based features (e.g., bytes per second)
[134]	flows	size, duration, inter-arrival time FFT value
[14]	flows	several volume-based and timing-based features autocorrelation coef.
[45]	flows	several volume-based and timing-based features, persistence
[150]	packets	several volume-based features, inter-arrival time
[50]	HTTP request-response pairs	sizes, inter-arrival times, durations, counts
[98]	packets (aggregated)	multiple aggregated statistics (means, variances,...)

Table 2.2: Summary of features and basic messages used for representation of network communication in the reviewed prior art.

3

Formal model of network communication

Works that employ various machine learning algorithms for analysis of network traffic data typically use a proprietary representation, often without deeper theoretical background. This often implies, that the given representation is not easily transferable to other scenarios and also not well suitable for changing a classifier, a detector or any other algorithm working on top of that representation. Alternatively, if the representation is easily portable to other domains, it is often thanks to its simplicity due to limited information used by the representation.

However, network communication can be observed and modeled at many levels and from various sources of input data. Therefore, one of the aims of this thesis is to provide a systematic view on the representation of the traffic, which is independent of specific scenario and can, thus, be easily adapted for different use-cases. Examples of different cases when it is desirable to model the traffic include sequences of packets or datagrams observed at the transport layer of the TCP/IP stack, NetFlow/IPFIX records, communication of end users with specific application layer servers (e.g., web servers) or even modeling of behavior of entire domains on the Internet. We therefore introduce an abstract model of communication which can be used to model any of the above mentioned cases. The representations described in further chapters can then be viewed as specific realizations of this abstract model. The basic idea of the approach presented in the thesis is to treat each communication as a set of observations which are realizations of a random variable with (mostly) unknown probability distribution. To describe models of communication independently of the the specific case, we introduce the term *message* as a basic unit of communication at the given level and the term *message set* which refers to the modeled communication. Below we formalize these entities of network communication:

Definition 1. Message is a basic unit of communication exchanged between two communicating peers. The exact definition of message is dependent on the level at which the communication is observed. It can be a single IP-layer or transport layer packet, a transport layer flow or even a request-response pair in a client-server communication. It is assumed that each message can be represented as a point

$$m \in \mathbb{R}^d$$

in an d-dimensional Euclidean space and is characterized by a set of d features (attributes). Attributes of a single message can be its size (e.g., number of bytes transferred in a packet), duration (if measurable, e.g., duration of a request-response loop in client-server communication), or time elapsed since the previous message was received (often called inter-arrival time). However, in general, the attributes can be any features observable and relevant for messages in the given scenario.

Definition 2. Message set is a set (or, possibly, a multiset[1]) of messages

$$\mathcal{R} = \{m_1, \ldots, m_n\}$$

sharing the same identifier of the communication. Again, the specific identifier of the message set is dependent on the level of observation and it can vary from a single flow specified by the both communicating endpoints to entire server specified by its IP address or hostname.

For example, if messages are defined as individual TCP packets, then a message set can be defined as one TCP flow of packets that share the same source and destination IP addresses and ports. Naturally, different message sets might have different cardinalities due to the different number of observed messages in them.

For purposes of a message set representation, each message $m \in \mathbb{R}^d$ in a message set $\mathcal{R}$ is treated as a realization of a d-dimensional random variable with a probability distribution $P_{\mathcal{R}} \in \mathcal{P}$, where $\mathcal{P}$ is the set of all probability distribution on the space of messages $\mathbb{R}^d$. The underlying assumption is that the message set $\mathcal{R}$ is fully determined by its distribution $P_{\mathcal{R}}$ followed by its messages. In practice, $P_{\mathcal{R}}$ is never precisely known, but it is observed through that finite set of messages:

$$\mathcal{R} = \left\{ m_i \in \mathbb{R}^d \mid i = 1, ..., n \right\}.$$

Designing of a proper representation and analysis of network communications is therefore transformed to the problem of proper representation and analysis of a probability distribution from a finite sample set of observations.

Definitions 1 and 2 are general and representations presented further were derived independently of any specific features observed on messages or the type of communication identifier. Thanks to this they can be easily adapted to different scenarios, algorithms working on top of them and sources of data.

As can be seen from the definitions above, the proposed formal model treats the messages as independent and identically distributed (i.i.d.) realizations of a random variable. Therefore, the model ignores order of the messages within a message set. While this might seem limiting, we are aware of this possible oversimplification and we use this approach for the following reasons:

1. In certain cases, the order of the messages is irrelevant and we want to cover also these cases. For example, when we aim at modeling of behavior of entire server which is contacted by many users concurrently and independently in random order, the order will be likely irrelevant (moreover, it can introduce unwanted artifacts if it is considered).
2. A logging system from which the messages are read might not be always able to guarantee the correct order of the messages or the underlying communication protocol might not support it (e.g., UDP protocol).
3. Considering the order of the messages would significantly increase the computational requirements on the representations which prevent their practical deployment on larger data.
4. Treating each message set as an i.i.d. sample allows to leverage the large variety of approaches to modeling of probability distributions that were mostly developed under the i.i.d. assumption.

[1] By a multiset we mean a set in which each value can occur more than once.

4

Datasets

In this chapter, we describe all the datasets that are in further chapters used for experimental evaluations. Network communication can be modeled at different layers and with different granularity. To reflect this, multiple datasets capturing various types of traffic were created during the work on this thesis and are summarized here.

For each dataset, we specify the definitions of messages and message sets in that particular case with respect to the formal model introduced in Chapter 3 and we also motivate the usage and purpose of each dataset.

4.1 Persistent connections

As we discussed in Section 2.2, persistent network communication (i.e., communication that appears repeatedly over time) can be found in many instances of malware. For example, the C&C channel of a bot malware is maintained throughout the life of the bot, and once it is lost, the control over the bot is lost, too. This condition implies that the channel needs to be persistent in the sense that the bot receives the commands repeatedly in time. However, bots are not the only type of malware which produces persistent communication. Malware can repeatedly check connection to the Internet, perform click fraud, or download advertisements all of which can manifest as a persistent communication.

On the other hand, legitimate activities of a user produce persistent communication as well. Repeated visits of a news portal, e-mail account, favorite social network or an application checking for updates are just few examples. All of these make the communication within a typical corporate network very heterogeneous and therefore provide good conditions for malware to hide its activities.

This dataset is composed of benign and malicious *persistent connections* that use the HTTP or HTTPS protocol to enable experiments focused on identification of malicious persistent connections in a network.

This datasets was created using web proxy logs from three companies, customers of Cisco Cognitive Threat Analytics [136], from one day of traffic in the year 2014. These companies are referred as A, B and C. A message in this dataset is defined as one request-response log entry logged by a proxy (e.g., a request for a particular HTTP resource). A message set is defined as one *connection* which is a set of web request-response pairs that share the same local and remote endpoints. The endpoints are identified by local user's username (local endpoint) and target second level domain (remote endpoint). Finally, a *persistent connection* is here defined as a connection in which requests occur repeatedly over a certain time period, e.g., one day. We used the modification [80] of the original approach [59] (see Chapter 2) to measure persistency of a connection and to identify the persistent ones.

To represent each web request-response pair (a message), following features are used:

1. **bytes sent** r_{up} from the client to the server,

2. **bytes received** r_{down} by the client from the server,
3. **duration**: r_{td} (in milliseconds) of handling the request (i.e., duration of the request-response loop),
4. **inter-arrival time** r_{ti} (in seconds) elapsed between start of the current and previous request.

Thus, a request-response pair is reduced to a 4-tuple $(r_{\text{up}}, r_{\text{down}}, r_{\text{td}}, r_{\text{ti}})$ which represents the message.

Totally, this dataset contains 3249 persistent connections (1732 in company A, 681 in company B, and 836 in company C). Because reliable manual labelling of all 3249 persistent connections in the dataset is difficult and subjective to human judgement, we treated all these persistent connections as legitimate (which is an acceptable assumption as they all come from monitored corporate networks). Malicious connections produced by malware were obtained from 14 different malware binaries executed in a malware laboratory. These binaries included variants of malware labelled by AV engines as ZeroAccess, Kelihos, ZBot, Asprox, Win32.Injector, Wapomi, Somoto, SS.Worm-generic and Downloader.UFN. From them, we isolated totally 50 persistent connections. Infection of a user by a malware sample was simulated during the experiments in which this dataset was involved by adding all the traffic from the malware sample's persistent connections to the background data. Because none of the domains utilized in malicious persistent connections were visited by any user in the background data, there was no risk of collision.

This dataset is used in experimental evaluation in Chapter 5.

4.2 Web servers

Many contemporary services have rather complex structure, as they are composed of several sub-services fulfilling specific tasks. The knowledge of this structure can simplify services' monitoring and improve security countermeasures against their misuses. However, the structure is not typically publicly known and its analysis by hand often includes reverse engineering of the communication protocol [38], which is time consuming and frequently too costly. Moreover, the rise of popularity of the encrypted HTTPS protocol, through which the data are often transferred, makes deep packet inspections difficult and also rules-out any port-based identifications.

The above described situation motivated the creation of this dataset which contains network communication belonging to web servers handling two widely used services, namely the Dropbox file storage and Windows Live platform. The purpose of this dataset is to enable experiments that aim at revealing structure of a service from its network behavior by assigning each server to the appropriate functional part of a service.

Similarly as the dataset of persistent connections described in Section 4.1, this dataset was created based on web proxy logs, too. Therefore, a message in this dataset is again one request-response log entry represented by the same four features (bytes sent, bytes received, duration and inter-arrival time). However, as in this case the services use the HTTPS protocol for communication, one request-response log entry might actually represent the entire HTTPS tunnel (see Section 2.1.1). A message set here is identified by a server's hostname. Hence, one message set aggregates all requests that were contacting the same server.

For the Dropbox service, the dataset captures network traffic of 188 servers under the `dropbox.com` domain from 5 days of traffic coming from a larger corporate network with approximately 10000 active users in the network and contains 17000 requests-response log entries per server on average. The data were collected during the year 2013. 95% of servers fell into 4 categories out of 11 identified in [38], namely: `clientX.dropbox.com` (meta data management), `dl-clientX.dropbox.com` (client storage), `dl-debugX.dropbox.com` (exceptions back-traces) and `notifyX.dropbox.com` (notifications about changes), where the letter `X` stands for one or more digits in the hostname.

For the Windows Live service, the dataset contains traffic belonging to 310 servers revealed under the `live.com` second level domain. The source data were the same as for the Dropbox service. However, unlike in the case of Dropbox, there exists no published ground truth for this service which would assign each server to its functional category.

This dataset was also used in experiments in Chapter 5.

4.3 Large scale HTTPS

In Chapter 2 (Section 2.1.1) we showed that information available for HTTPS connections — because of the encryption — is very limited in the logs produced by web proxies. This, collaterally with the trend of increasing the total portion of web traffic which is using HTTPS, makes the identification of HTTPS connections related to malicious activity a very pressing research problem. This motivated formation of this dataset which is created purely from log entries of HTTPS tunnels.

One message in this dataset is defined as one log entry containing information about one HTTPS tunnel (log entries of unencrypted HTTP requests were ignored for purpose of building this dataset). Representation of one message is similar as in the datasets described above in Sections 4.1 and 4.2. However, we emphasize that the encrypted tunnels may be in fact composed of multiple request-response pairs for which the volumetric information was aggregated by the proxy when the log entry was created. The features that represent one message are therefore:

1. **bytes sent** through the tunnel from a client's machine to a target server,
2. **bytes received** through the tunnel by a client's machine from a server,
3. **duration** of the tunnel, i.e., length of the time interval for which the tunnel was active,
4. **inter-arrival time** (in seconds) elapsed between two consecutive requests for establishing a tunnel from a client't machine to the same server. The inter-arrival times are computed separately for each server visited by the given client, independently of other servers. For example, if a client is repeatedly contacting `google.com` and `yahoo.com` domains, then the 4-tuples representing tunnels to `google.com` will contain values of inter-arrival time between the consecutive tunnels established to `google.com`, independently of tunnels established to `yahoo.com` (that might be interleaved with them).

One message set in this dataset spans all messages originating from the same client within a 5-minute window. Therefore, it contains the entire HTTPS communication of one client in the given time window. The motivation for modeling of the whole client machine is to capture the entire context in the message set. Malware infections might not use only one isolated connection to perform all their activities. Instead, the malware can, for example, test availability of Internet connection, get information about the timezone or geolocation of the infected user, then it may start trying to contact one or more of its command and control servers. These all might be weak indicators of the malware's presence that influence the overall profile of the infected client's behavior and can be contained in a message set.

In order to enable experiments on large scale data, this dataset was created from logs reported by web proxies during the period of one day from 500 corporate networks that were using Cisco's Cognitive Threat Analytics [136] cloud solution. Besides the features of the HTTPS tunnels, the proxies were able to log also SHA hash of the process' binary that initiated the given HTTPS connection. These hashes were used to annotate the dataset. Specifically, a process binary was considered malicious if its corresponding process was marked as malware by at least 20 anti-virus engines used by the Virustotal[1] service. Subsequently, these labels were propagated to individual messages (the HTTPS tunnels) — a message was labelled as malicious if it originated from a binary which was labelled as malicious. Otherwise,

[1] `virustotal.com`

a message was considered legitimate. Finally, a whole message set was labelled as malicious if it contained at least one malicious message.

In total, the dataset contains 145 822 799 messages to 475 605 unique servers which result in 8 642 368 unique message sets. 44 213 message sets were labelled as malicious. The total volume of data transferred by the messages was approximately 10 TB.

This dataset is utilized in Chapter 6 for the experiments on large scale similarity search.

4.4 TCP flows

This dataset contains logs of flows of TCP packets collected during one work day in February 2015 in a corporate network of a company with 550 active users with computers running mostly Windows or OSX (macOS) operating systems. To access internal network and the Internet, these users used the AnyConnect VPN client [29] modified such that it exported names of the processes running on users' machines that generated the network communication (this feature is not present in the standard AnyConnect VPN client and the users were informed about the extension). The modified client collected following information about each flow of TCP packets: name of the process (and its hash) which generated the flow, local user IP address, remote server IP address, remote server port, number of bytes and packets sent to and received from the server, and duration of the particular flow. Additionally, since the AnyConnect client is able to see also packets at the transport layer within each flow, it was possible to log the sequence of their sizes and inter-arrival times for each flow. An example of collected data for a single TCP flow (in JSON format as provided by the logging software) is shown in Figure 4.1.

```
{"flow" :
{"an" : "APSDaemon.exe",
"t_start" : 1424178205, "t_end" : 1424178207,
"sa" : "192.168.2.105" , "sp" : 59453,
"da" : "80.239.137.24", "dp" : 80,
"user" : "john.smith",
"packets" : [
{"dir" : ">", "b" : 138, "ipt" : 0}
{"dir" : ">", "b" : 115, "ipt" : 50}
,{"dir" : "<", "b" : 268, "ipt" : 0}
,{"dir" : "<", "b" : 1336, "ipt" : 248}
] }}
```

Fig. 4.1: Data of a single TCP flow collected by the modified AnyConnect client (in JSON format), including the application label (the "an" field).

The data collected by the AnyConnect client thus allowed to define the message either:

- as a TCP packet at the transport layer in which case a single TCP flow corresponds to one message set. Tthis dataset is further called Any-P.
- or as an entire TCP flow in which case one message set corresponds to all TCP flows interchanged between the client and the server endpoint, with identifier of a message set defined as the triplet `[client IP, server IP, server port]`, this dataset is further called Any-F.

Differences in both definitions are outlined in Figure 4.2. Each message at the transport layer (in the Any-P dataset) is represented by its size (in number of bytes), direction and time elapsed from its predecessor (i.e., the inter-arrival time). Similarly as in work [71], the direction of each message is encoded by the sign of its size and inter-arrival time. In the

Any-F dataset, each message is represented by the number of bytes and packets sent to and received from the server, and by the duration of the flow.

Since the modified AnyConnect Client included names of clients' processes initiating the connection, this information was used as ground-truth labels for message sets of this dataset. Both datasets contain message sets belonging to 69 applications, with 198 786 flows in total. The number of flows per application varies from 550 to 9800. A complete list of application names can be found in Appendix.

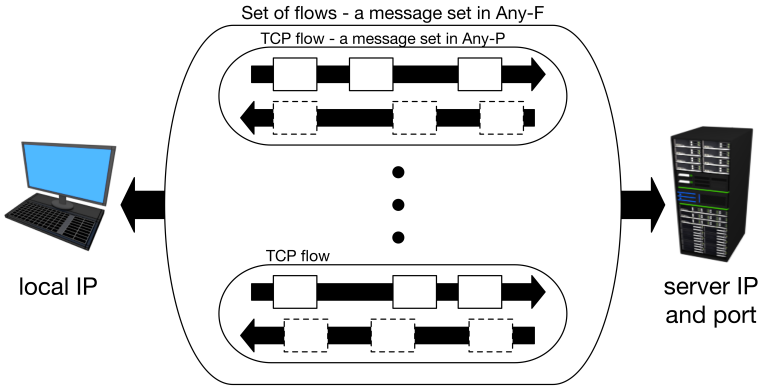


Fig. 4.2: Two definitions of messages in the *TCP flows* dataset. Any-P dataset defines a message as one packet at the transport layer (solid and dashed rectangles), while a message set is defined as one TCP flow of packets. Any-F dataset defines a message as one TCP flow and a message set as a set of all TCP flows interchanged between a client and a server.

This dataset is used for experiments in Chapter 7, where it is also used for comparison of the representations proposed in this thesis and to prior art methods, because due to data retention policies, the older datasets *Persistent connections* and *Web servers* were not available to us at the time of designing the representation described in Chapter 7. Additionally, this dataset allowed to perform the comparison at different levels — at the level of TCP packets (Any-P) and at the level of TCP flows (Any-F).

4.5 Hostnames

The motivation for creating this dataset was to enable experiments that test identification of operating systems running on network devices based on observing only web servers that the devices contact.

One message in this dataset is a pair of contacted web server's hostname and a timestamp indicating when the given contact was observed. A message set is defined as communication belonging to one network device (e.g., a PC, laptop or cell phone) in the entire time period from which the dataset was created. Each message set is labelled by the operating system family that the given device was running. The dataset contains devices running three main operating system families that are most prevalent across corporate networks which are: Apple devices (running iOS/macOS/OSX operating systems), Android devices (i.e., mostly mobile devices running Android-based operating systems) and Windows devices (running Microsoft Windows-based operating systems).

The dataset was created from web proxy logs coming from three larger companies and covers approximately 28 hours of traffic (collected in February 2019). The prevalence of individual operating system families in these companies is summarized in Table 4.1.

The dataset is used for the testing the representation proposed in Chapter 8

	Android	Apple	Windows
Company 1	2394	2431	30986
Company 2	490	2710	23459
Company 3	1477	3043	24713

Table 4.1: Prevalence (numbers of unique devices) of the three operating system families in companies from the *Hostnames* dataset.

5

Histogram representations

Histograms are a widely adopted method to capture distribution of an observed random variable in many domains. Let's have an univariate real random variable f with unknown probability distribution observed through a set of realizations $\mathcal{R} = \{f_1, ..., f_r\} \subseteq \mathbb{R}$ and so called histogram bins $b_1, ..., b_m \in \mathbb{R}$. A histogram capturing its distribution can be viewed as an m-dimensional real vector $h \in \mathbb{R}^m$ in which the i-th element h_i represents the number of elements from $\mathcal{R}$ that are closest to the i-th bin b_i of the histogram among all bins $b_1, ..., b_m$. The set of histogram's bins is determined beforehand. A common practice is that the bins are centered equidistantly to cover the domain of the variable f. Nevertheless, equidistant bins are not always necessary and, in general, bins' centers $b_1, ..., b_m$ can be placed arbitrarily to cover the interesting regions of the target domain to balance accuracy of the produced representations and sizes (number of elements) of the histograms ([142, 123]). However, in the domain of network communication representation, we use the equidistant bins because we do not assume that anything is known about the modeled distribution beforehand (in practical deployments the histogram will be often built on-line as new observations arrive).

To illustrate the histogram representation by an example, let us consider that 5 bins are centered equidistantly at values 1, 2, ... ,5 and we collected 6 observations of the variable f such that

$$\mathcal{R} = \{1.3, 2.3, 2.4, 1.8, 3.2, 2.1\}.$$

Then, the histogram h will be a 5-dimensional vector with following values:

$$h = (1, 4, 1, 0, 0)$$

Here we can see that most of the observations fell into the bin b_2 which is centered at value 2. Specifically, the observations 2.3, 2.4, 1.8 and 2.1 contributed to that bin. Therefore, the value of the element h_2 in the histogram h is 4. In case of the other bins, the principle is analogous.

A histogram h can be L_1-normalized to capture relative frequencies. This is done by multiplying all its elements by $\frac{1}{|\mathcal{R}|}$. The normalization assures that the sum of all histogram's elements is 1 and the relative frequency at the i-th element of h can be interpreted as an estimation of probability that the next realization of f will be close to the center of bin b_i. Naturally, histograms can be extended to capture distributions of multivariate d-dimensional random variables with bins $b_1, ..., b_m \in \mathbb{R}^d$.

5.1 Soft histograms

Commonly used approach to build a histogram, which we call *hard* histogram here to distinguish it from the studied modification, is to use each observation to update only the one bin into which the given sample falls (i.e., the closest bin). For example, in a one-dimensional case, the i-th bin with bounds $[b_i, b_{i+1})$ is updated by 1 irrespectively if the sample is close

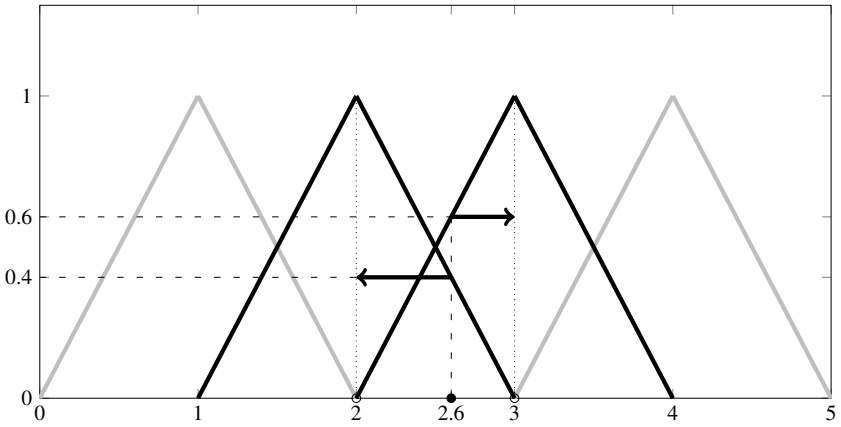


Fig. 5.1: Example of updating a one-dimensional soft histogram with value 2.6. It contributes with 0.4 to the bin centered in 2 and with 0.6 to the bin centered in 3. Filters that influence the contribution are highlighted.

to b_i or b_{i+1}, as can be seen in the example above (consider values 1.8 and 2.4 that both contribute to the bin b_2 centered at 2). This strict quantization makes values of histogram bins relatively sensitive to small variations and noise in the data. In the domains of image and signal processing [48, 112], this sensitivity is removed by using so-called *soft* histograms, where each sample updates two (in the one-dimensional case) closest bins by values proportional to their distance. In the simplest form, the sample's contribution to two nearest bins depends linearly on distance to them, which corresponds to the triangular filters used in signal processing [112]. When a soft histogram is updated by a new observed value u, the two nearest bins with centers in $\lfloor u \rfloor$ and $\lfloor u \rfloor + 1$ are updated by $1 - (u - \lfloor u \rfloor)$ and $u - \lfloor u \rfloor$, respectively. The situation is depicted in Figure 5.1, where a one-dimensional histogram is updated by a sample $u = 2.6$. Its two nearest bins centered in $\lfloor u \rfloor = 2$ and $\lfloor u \rfloor + 1 = 3$ are updated by $1 - (u - \lfloor u \rfloor) = 0.4$ and $u - \lfloor u \rfloor = 0.6$, respectively.

To capture joint frequencies of multiple quantities, the soft histogram is naturally extended to multiple dimensions. Without loss of generality, it is assumed that a d-dimensional soft histogram has equidistant bins centered at integer lattice points $b_i \in \{0, ..., n\}^d$, where n acts as an upper bound in each dimension on values to be inserted to the histogram. Updating a soft histogram with a tuple $(u_1, u_2, \ldots, u_d)$ then means first calculating indices l_i and contributions v_i to "left" bins as

$$l_i = \lfloor u_i \rfloor, \; v_i = 1 - (u_i - l_i),$$

and then updating all bins centered in vertices

$$\{(l_1 + i_1, \ldots, l_d + i_d) | (i_1, \ldots, i_d) \in \{0, 1\}^d\}$$

with values

$$\prod_{j=1}^{d} v_j^{1-i_j} (1 - v_j)^{i_j}.$$

The histogram construction is finalized by mapping the histogram to an $(n+1)^d$-dimensional column vector in order to simplify further processing. This way the soft histograms are able to estimate the underlying probability density in more accurate way than the commonly used hard histograms.

Additionally, the term frequency - inverse document frequency (TF-IDF) weighting is applied on the histograms, which puts more emphasis on less frequent non-zero items. It multiplies each bin b_i in all histograms by $\log\left(\frac{N+1}{N_{b_i}+1}\right)$, where N is the total number of

histograms and N_{b_i} is the number of histograms with non-zero value of the bin b_i. TF-IDF weighting is very common in fields utilizing sparse representations, such as text documents analysis [78] or computer vision [27].

An advantage of the joint histogram is that it captures dependencies between the quantities. Its disadvantage is that in order to capture the dependencies accurately, the number of bins can be high. However, we argue that the total dimension of the histogram is not a problem because as shown below in the experiments, the histogram will be typically very sparse. Moreover, we note that, for example, the field of text document analysis [119] works with high dimensional yet sparse data as well.

Histograms are therefore a straightforward way how to represent the message sets based on messages as individual observations according to the definitions in Chapter 3. The histograms provide a method how to represent each message set as a real vector that can be consumed by other algorithms and compared to each other using similarity functions applicable to histograms. In further text, we refer to the histogram representations of message sets as *fingerprints*.

5.2 Experimental evaluation

In the following experiments, we present two different scenarios in which we verify that the soft histogram representations improve the results of traffic analysis. First, we show that the soft histograms improve performance of an outlier detector focused on detection of malicious persistent connections. Second, we demonstrate that the soft histograms are able to improve unsupervised discovery of a web service's structure. Furthermore, in both experiments, we show that the histogram-based representations are very sparse and can be thus effectively stored.

5.2.1 Detection of malicious persistent connections

In this experiment, we study outlier detection run on top of the histogram representations and show that the soft histograms are able to improve its results. For this experiment, we use the dataset called *Persistent connections* described in Section 4.1.

Representations of persistent connections — their fingerprints — are created by using tuples representing all messages belonging to the given connection to build a joint histogram. In order to narrow the range of modeled values and to equalize variances on low and high values, all values in tuples are transformed to logarithmic scale before calculating the joint histogram. Based on values observed in the data, we set $n = 10$ to be sufficient implying the total number of bins to be $11^4 = 14641$. The fingerprint of a connection is therefore a joint histogram of the four quantities estimated from all tuples from a given persistent connection.

We additionally use a limit K_u to filter out persistent connections to remote endpoints that were contacted by more than K_u local users. The reason is that these domains are expected to be very popular services and therefore not serving malicious purposes (e.g., Google). For such popular services, it would be possible to build a special model to identify outliers within them, which can be an indication of service misuse. Based on the total volume of traffic in the dataset, we set the limit $K_u = 10$.

Before diving into the outlier detection, we verify that the soft histogram representations are indeed sparse. This is illustrated in Figure 5.2 which shows the cumulative distribution of non-zero bins in histograms of malicious and background persistent connections from the dataset used. By background connections we mean all connections coming from the three companies in the dataset. The fingerprints of malicious connections were computed from all 50 persistent connections in the dataset that belong to malware. We can see that more than 80% of fingerprints have less than 250 non-zero bins out of 14641. This confirms that the soft histogram representations are very sparse and can be effectively stored. The curve for

fingerprints of malware is even steeper which means that the number of non-zero bins in histograms is lower and the connections are more regular. This confirms that malware has more regular behavior as the histogram fingerprints have lower entropy.

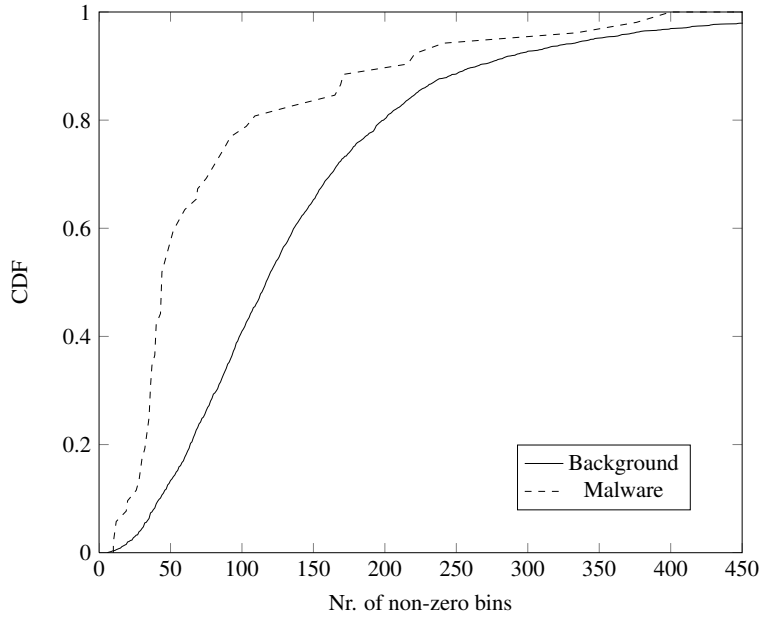


Fig. 5.2: Cumulative distribution functions (CDF) of number of non-zero bins in soft histograms of background (i.e., assumed to be mostly benign) and malicious persistent connections.

Outlier detection

In this part of the experiment, we compare the performance of the traditional hard histograms and the proposed soft histograms in outlier detection setup which is employed to identify the malicious persistent connections.

The histogram representations of persistent connections are numerical vectors of fixed length on which most of outlier detection algorithms can be readily applied [1]. Since we can expect that malware's fingerprints form small clusters (because multiple connections of malware can have very similar properties), we decided to use primarily the OutRank [94] algorithm, because it was designed to be robust against cases when outliers form small clusters (for details, see Appendix). The formation of small clusters can be caused, for example, by multiple infections of the same malware family with similar behavioral patterns or by a single malware instance maintaining several persistent connections with the same purpose (to improve robustness of the channel).

Together with the OutRank algorithm, we test the histogram fingerprints also with the k-nearest neighbors (k-NN) based detector [126], which is an asymptotically consistent density level estimator. Therefore, the k-NN based detector would provide good performance if each of the malicious connections would be an isolated outlier very dissimilar to others.

The core idea behind the OutRank algorithm is similar to that behind the famous PageRank [104]. The outlier score assigned to each object in the dataset is based on random walks in a similarity graph which is built to represent the dataset. Nodes of the similarity graph are the individual objects and weighted edges between them represent their pair-wise similarities. By converting the weights of the edges into transition probabilities, i.e. probabilities of moving from one node to an adjacent one during a random walk in the graph, a transition

matrix for the entire graph is obtained. Using this transition probability matrix, its dominant eigenvector is computed which determines the outlier scores for individual nodes, i.e. for the objects in the dataset. The rationale behind this approach is that if we consider all possible paths in the similarity graph, nodes that are outliers (connected with the majority of the remaining nodes via paths with low transition probabilities) will be visited only rarely by the random walks. This is valid even if the outlying nodes form small internally connected clusters, because these clusters will be still separated from the majority by the low probability paths.

In case of the k-NN detector, the outlier score for each object is assigned based on the average distance to its k nearest objects in the dataset. Therefore, it directly aims to identify objects that lay in a region with very low density of other objects. Therefore, it exactly stems on the very basic definition of an outlier which states that "an outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism" [70]. However, this approach is sensitive to cases when the outliers might form small clusters, because in such cases the average distance to the nearest objects is significantly influenced.

The main difference in the OutRank and the k-NN approaches is therefore in the *global* versus the *local* view of the detectors. While OutRank assigns outlier scores using the global properties of the dataset, the k-NN paradigm considers the local properties induced by those k nearest neighbors of the object in question (which, on the other hand, might be convenient in certain cases as we will see in further chapters of this thesis).

The choice of distance measure between two histograms which is used by the outlier detection algorithms was inspired by measuring similarity of vectorized text documents, because in analogy to them, histogram fingerprints are sparse. We therefore use cosine distance defined as

$$d(h_1, h_2) = 1 - \frac{h_1^{\mathrm{T}} h_2}{||h_1|| \cdot ||h_2||},$$

where h_1, h_2 are histogram fingerprints of two persistent connections that are being compared.

The quality of detection was measured by the area under the ROC (receiver operating characteristic) curve (AUC) [44], which is a common measure in cases when detection threshold can not be determined beforehand. AUC was calculated for every combination of background data set (i.e., for each of the three companies A, B and C) and each malware sample (out of 14), which lead to 3×14 evaluations of every detector. Average AUCs over all 14 malware samples for combinations of outlier detection algorithm and histogram types are presented in Table 5.1. We can see that in all cases detectors employing representations based on soft histograms performed better irrespectively to the used outlier detection algorithm. This confirms that the soft histograms indeed contribute to improved quality of the detection independently of the outlier detection method.

	hard histograms		soft histograms	
Data set	k-NN	OutRank	k-NN	OutRank
A	0.855	0.892	0.862	**0.942**
B	0.889	0.896	0.912	**0.935**
C	0.871	0.867	0.880	**0.933**

Table 5.1: Average AUC values (computed over the 14 malware samples) for all combinations of hard and soft histograms, and OutRank and k-NN outlier detection algorithms. Higher is better with one being maximum, best results on each company are bold-faced.

To get further insight into the detections we analyzed 10 persistent connections with the highest outlier scores assigned by the best detector from each data set. Deeper analysis of these most anomalous connections revealed 3 persistent connections in data set A, 1

connection in data set B and 2 connections in data set C that were provably related to malicious activity. The most frequently observed type of possibly legitimate but anomalous connections were those trying to reach a web server which was either unavailable (returning response code 50X) or the requested resource could not be accessed.

We can conclude that the soft histogram representations of messages sets are a suitable way how to capture behavioral patterns in persistent connections using very lightweight features. The results achieved in outlier detection provides solid basis for using this approach in intrusion detection systems. The performance measured by means of the AUC is sufficient for including this detector in an ensemble of detection methods within a more complex system [63].

5.2.2 Servers clustering

In this experiment, we study the improvement gained from using the soft histogram in a clustering algorithm applied to identify groups of similarly behaving servers. The dataset used for this experiment is the *Web servers* dataset described in Section 4.2. We use separately both services that are contained in the dataset — the Dropbox service for which the ground truth is available, and the Windows Live platform.

Because the messages in the *Web servers* dataset are 4-tuples of the same type as in the *Persistent connections* dataset used above, the representation of servers (their histogram fingerprints) are built in the same way as fingerprints of persistent connections.

First, we again verify that the soft histogram fingerprints are sparse even in the case of servers' representations. This is illustrated in Figure 5.3 which shows the cumulative distribution function of the number of non-zero bins in fingerprints of all servers in the dataset. We can see that fingerprints of more than 95% servers have less than 5% of non-zero bins. Besides demonstrating the sparsity of the representation, it also verifies that the behavior of servers is very regular. This provides good dispositions for identification of servers' functionality.

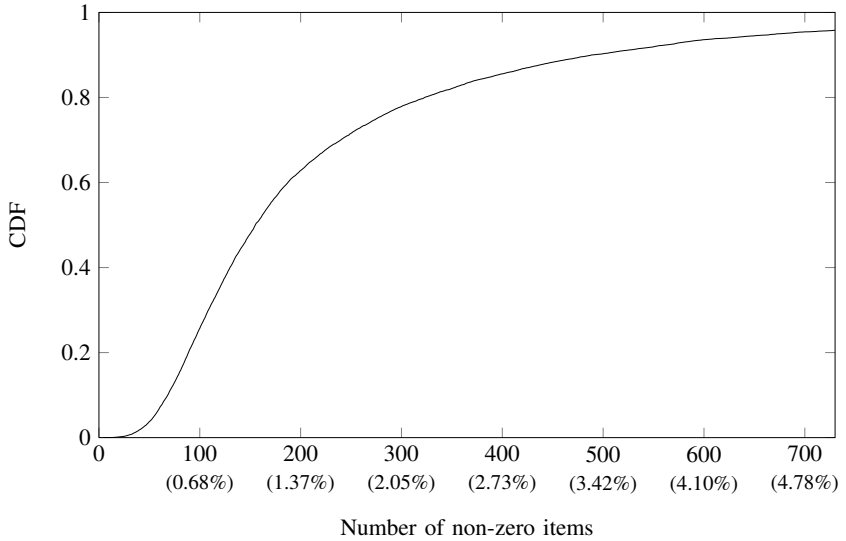


Fig. 5.3: Cumulative distribution function (CDF) of the number of non-zero bins in fingerprints of all servers in the *Web servers* dataset. The CDF plot demonstrates that fingerprints are typically very sparse — the total number of bins in a fingerprint was set to $11^4 = 14641$, thus more than 95% of fingerprints have less than 5% of non-zero bins.

Clustering of servers' fingerprints in order to discover the functional groups follows the usual clustering steps. First, similarities between all fingerprint pairs are calculated. Then, optionally τ_s smallest similarities are set to zero to accentuate true clusters, and finally the chosen clustering algorithm is applied. The steps are detailed below.

Similarity measure

While in the outlier detection we employed cosine similarity to compare two fingerprints, in this experiment we evaluated two similarity measures between fingerprints h_1 and h_2. s_c is the previously used cosine similarity while s_e is based on Euclidean L_2 distance scaled to $[0, 1]$ such that both similarity functions have the same range. The scaling leverages the fact that the upper-bound on L_2 distance between h_1 and h_2 is $\sqrt{2}$, because L_1 norms of the normalized fingerprints are 1 and their items are non-negative (for purpose of computing the similarity measures, the fingerprints are L_1-normalized). These similarities are formally defined as

$$s_c(h_1, h_2) = \frac{h_1^T h_2}{||h_1||_2 \cdot ||h_2||_2} \tag{5.1}$$

$$s_e(h_1, h_2) = \frac{\sqrt{2} - ||h_1 - h_2||_2}{\sqrt{2}} \tag{5.2}$$

Discarding low similarities

τ_s percent of the lowest similarities are set to zero making the respective servers completely dissimilar. Although this filtering might decrease noise and accentuate the true clusters, it can also discard too much information rendering the true clusters unrecognizable. The impact of this filtering on the accuracy of clustering is investigated further as part of this experiment.

Clustering algorithm

In general, any clustering algorithm accepting either feature vectors from $\mathbb{R}^d$ or a similarity matrix of the clustered objects can be used ([146, 100]). The clustering algorithm primarily chosen for this experiment is the Louvain method [16] designed for discovering communities in graphs.

The nodes of the graph on which the Louvain clustering is applied represent servers' fingerprints while the similarity measures s_c or s_e determine weights of the edges between the nodes. The filtering described in the previous paragraph therefore has an effect of removing edges with low weights. An advantage of the Louvain method is the optimization of the number of clusters, which is useful for applications when the desired number of clusters is not known beforehand. While the representation of the data by a similarity graph might seem counterintuitive, it proved to be very helpful in a similar task of discovering groups of malicious servers in [81]. Additionally, it allows for easy visualizations of the data even in case of non-metric spaces.

The evaluation of the clustering is primarily done on servers belonging to the Dropbox service, because the ground truth exists for these servers. Additionally, we show the results of clustering of servers that belong to the Windows Live platform to illustrate that the method can be used for analysis on various types of services.

Dropbox analysis

For purposes of the evaluation, we refer to the four groups of Dropbox servers that are in the dataset labeled as different functional parts as *ground truth groups*.

The quality of produced clustering solutions was measured by the Adjusted Rand Index (ARI) ([114, 74]) measure, which is a general measure taking value in the range $[-1, +1]$ evaluating agreement of two clustering solutions (higher value means better match of the

solutions). In our case, we always compared outcome of an evaluated clustering method to the ground truth groups.

The experimental results of comparison between the hard histograms and the soft histograms are shown in Table 5.2. The table compares the histograms by means of ARI of the produced clustering solutions in different settings specified by the similarity function, filtering level τ_s and a clustering algorithm. Besides the Louvian clustering algorithm we include the spectral clustering [100] which is closely related to the k-means algorithm but is applicable also in non-metric spaces.

The results show superiority of the soft histograms over the traditional hard ones for any combination of similarity measure, filtering level and clustering algorithm.

Histograms	Clustering	Similarity	Filtering level τ_s 0%	20%	50%	70%
soft	Louv.	Cosine	0.947	0.947	0.947	0.443
		Euclidean	0.723	0.732	0.732	0.948
	Spect.	Cosine	0.887	0.853	0.860	0.788
		Euclidean	0.793	0.582	0.751	0.746
hard	Louv.	Cosine	0.740	0.945	0.441	0.310
		Euclidean	0.722	0.723	0.723	0.933
	Spect.	Cosine	0.849	0.822	0.138	0.035
		Euclidean	0.786	0.489	0.653	0.699

Table 5.2: Values of Adjusted Rand Index (ARI) comparing agreement between the ground truth clustering of the Dropbox servers and the clustering produced by the evaluated combinations of histogram type, clustering algorithm, similarity measure and filtering level (higher is better, minimum is -1, maximum is 1).

To illustrate how well the obtained clusters represent functional parts of Dropbox, Figure 5.4 shows empirical estimates of marginal probability distributions of all four modeled quantities. The distributions were estimated from soft histogram fingerprints in four largest clusters obtained by the Louvain clustering with cosine similarity and $\tau_s = 0\%$. Titles of the groups correspond to the most dominant part of Dropbox service in the cluster.

We can see that requests to `notifyX` servers have zero uploaded bytes, nearly constant number of downloaded bytes, they are periodic and of the same (relatively long) duration. This well corresponds with the notify servers informing clients about changes by implementing a push mechanism. Contrary to that, connections to `dl-clientX` are aperiodic with relatively large amounts of uploaded and downloaded bytes. This is caused by the fact that these servers are used to upload and download content to/from the storage. Servers in `dl-debugX` are used to send debug data and requests to them have long durations like requests to `notifyX`, but unlike those they have non-zero uploaded bytes and are aperiodic. The higher volume of transmitted bytes can be caused by the long duration of requests. Finally, requests to `clientX` have the shortest durations among all groups and in comparison to requests to `dl-debugX` and `dl-clientX` the amount of transferred bytes is smaller. According to [38], these servers handle meta-data (but the exact protocol is not discussed in the work).

To verify quality of the clusters, Table 5.3 shows clusters' purities and recalls. Purity of a cluster is defined as the ratio of dominant ground truth labels within the cluster to its size. Similarly, the recall is defined as the ratio of the dominant ground truth labels within the

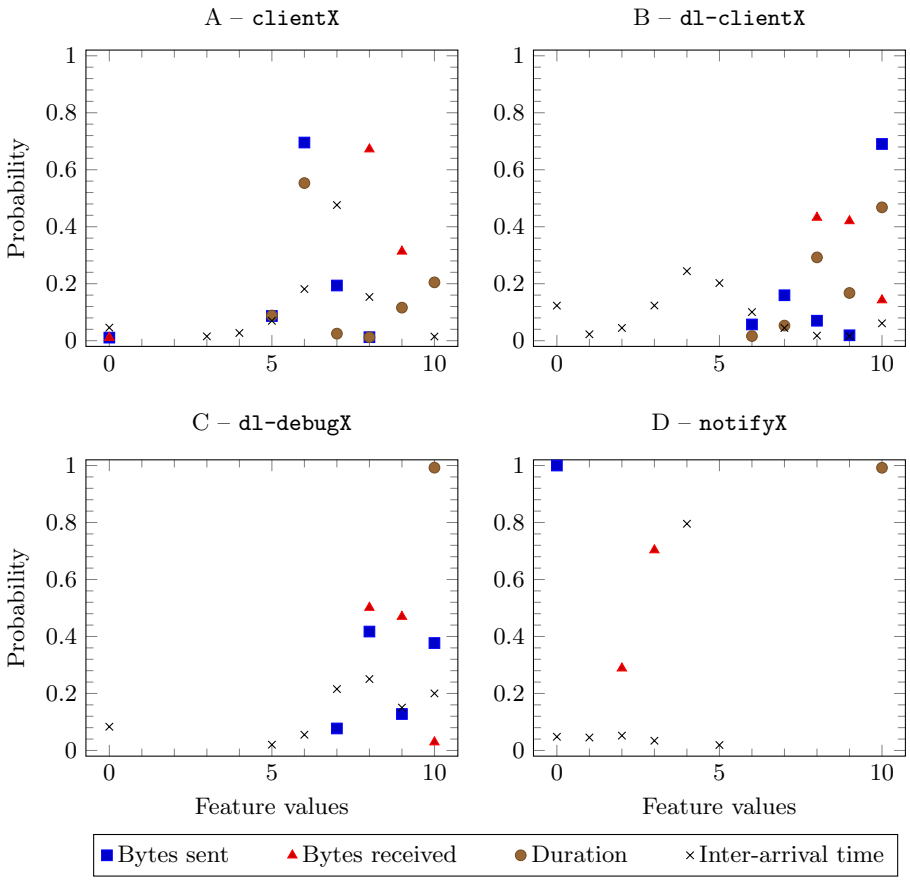


Fig. 5.4: Marginal probability distributions of the four observed features, estimated from fingerprints of Dropbox servers found in the four largest clusters.

Cluster	Ground truth group	Purity	Recall
A	`clientX`	1.00	1.00
B	`dl-clientX`	0.86	1.00
C	`dl-debugX`	1.00	1.00
D	`notifyX`	1.00	1.00

Table 5.3: Dominant types of Dropbox servers in the four largest clusters.

cluster to the total number of those labels in the entire dataset. We can see that all clusters are pure and servers with the same functionality are in one cluster (with the only exception of cluster B (`dl-clientX`) which contains some servers from the remaining 5% of servers providing other Dropbox functionality).

A similarity graph (using the cosine similarity function) of Dropbox servers is visualized in Figure 5.5 with the help of Gephi [9] tool and the Force Atlas 2 drawing algorithm [76]. This algorithm attracts graph nodes with high similarities towards each other while separating pairs of nodes with low similarities. The graph supports conclusions drawn above — despite that marginal distributions of the features for the `dl-debugX` (blue) and `dl-clientX` (red) servers are more similar, nodes representing servers from the same ground truth groups are attracted together because of high similarities of servers inside both of the ground truth

groups. Therefore, the clustering algorithm using the soft histograms is able to separate these servers correctly.

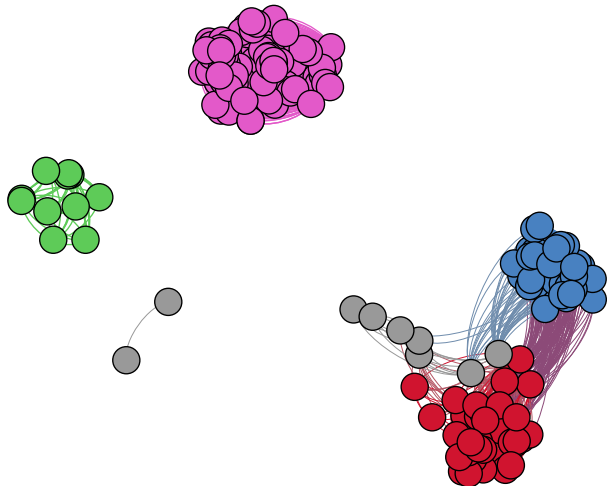


Fig. 5.5: Cosine similarity graph for the Dropbox servers. Servers from the four ground truth groups are highlighted by different colors: `clientX` servers are purple, `dl-clientX` servers are red, `dl-debugX` are blue, and `notifyX` servers are green. The remaining 5% of servers are grey.

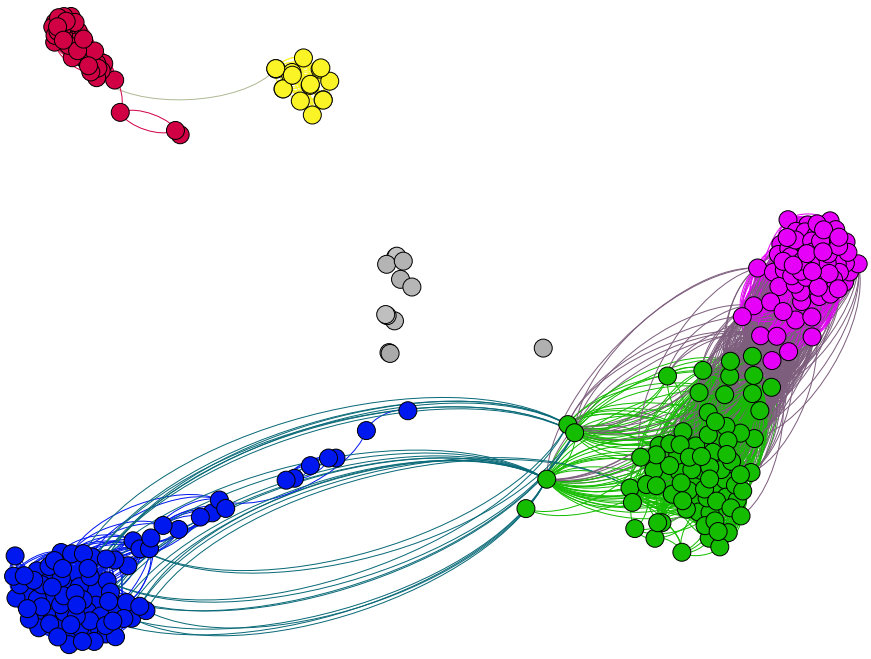


Fig. 5.6: Cosine similarity graph for the servers belonging to Windows Live platform. Five largest clusters discovered by the Louvain method are highlighted with color, the red cluster and the blue cluster contain servers probably involved in mail-related services while the green cluster and the purple cluster contain servers that handle instant messaging. The yellow cluster gathers servers related to storage services. The remaining nodes (6%) are grey.

Windows Live analysis

The ability to distinguish servers hosting different applications within the Windows Live service is demonstrated in Figure 5.6. The figure shows a cosine similarity graph of Windows Live servers from the *Web servers* dataset. Again, the graph was visualized with help of the Force Atlas 2 algorithm and five largest clusters (containing 94% of all servers) discovered by the Louvain method are highlighted with different colors (red, blue, green, purple, and yellow). For these servers, we did not have the exact ground truth like in the case of Dropbox, thus, the detailed analysis of the method's performance could not be performed. However, as shown in Figure 5.6, the servers form several well-shaped clusters which suggests that these clusters likely gather servers running different applications. Indeed, the analysis of hostnames belonging to servers in the five largest clusters revealed that 76% of servers in the blue and in the red cluster have a word "`mail`" in their hostnames. This might be an evidence that these clusters likely represent servers engaged in mail services of the Windows Live platform. On the other hand, 99% of servers in the green cluster and in the purple cluster contain a word "`messenger`" in their hostnames. This points out that these two clusters gather servers handling the instant messaging service (Windows Live Messenger). Similarly, 67% of servers in the yellow cluster have a "`storage`" substring in their hostnames. Thus, we can assume that the servers from the yellow cluster are involved in storing users' data.

5.3 Chapter summary

In this chapter, we studied a method for representation of network communication which is based on empirical histograms. The proposed representation uses so called soft histograms, which apply triangular filters to distribute contribution of each observation among multiple bins. This helps to suppress noise in the observations and, as proved by the experiments, brings significant improvement in quality of results of algorithms working on top of this representation.

In the experimental evaluation we showed that the histogram representations can be successfully used in tasks of unsupervised analysis of network communication — the outlier detection and clustering of servers to identify groups that provide the same type of service. In both cases the soft histograms showed that they are able to capture important behavioral patterns in the traffic at different levels — either at the level of individual connections or at the level of entire servers — and they provide significantly better performance compared to traditional hard histograms.

6

Large scale processing of histograms

The last chapter demonstrated that fingerprints based on soft histograms are successfully used in analysis of network communication. Thanks to their sparsity, these representations are very economic and can be efficiently processed. Hence, computational requirements are not a primary concern. However, the volume of network communication has been constantly rising [132], and analytical systems have to face scalability issues even in case of such economic representations. Furthermore, data from multiple networks are often aggregated in order to extend the knowledgebase of malware [81, 79] which again increases the amount of data that needs to be processed by one system.

This chapter is therefore devoted to construction of a pipeline which enables analysis of soft histogram fingerprints on large volumes of data. Specifically, we are focused on designing an effective similarity search method which is able to perform high number of k nearest neighbors search queries on a large database of histogram fingerprints. This is motivated by the properties of data visualized in Figure 6.1. The figure shows a t-SNE [137] similarity plot of fingerprints representing message sets from the *Large scale HTTPS* dataset described in Chapter 4 (Section 4.3).

As we can see in the figure, malicious fingerprints tend to be grouped together in many smaller clusters. Therefore, using a large database of annotated fingerprints which can be searched for fingerprints that are most similar to an unknown one provides a good way how to analyze new fingerprints. This is in fact similar to the situation which we faced in outlier detection in Chapter 5. While in the outlier detection setup this formation of clusters decreased performance of the detector based on k-NN search, it can be leveraged here to use the k-NN queries on known data to describe the unknown.

In a straightforward implementation, the k-NN search is computationally very demanding because it computes pair-wise distances among all objects in the database. However, if we leverage properties of metric spaces to organize the data, unnecessary comparisons of objects can be avoided. If we additionally allow approximate search results (which does not guarantee that exactly the nearest neighbors will be always returned), the k-NN search can be successfully implemented in a MapReduce programming model allowing its deployment on large databases of fingerprints.

The structure of this chapter is as follows: We introduce the problem of similarity search on the histogram data (Section 6.1). Then, in Section 6.2 we propose a method for effective k-NN similarity search and discuss the optimizations used. In Section 6.3, we present a parallelized implementation of the similarity search that uses the MapReduce programming model. Finally, we perform experimental evaluation of the proposed method on the dataset *Large scale HTTPS* in Section 6.4, showing that the large scale similarity search can be used to detect malware in HTTPS traffic.

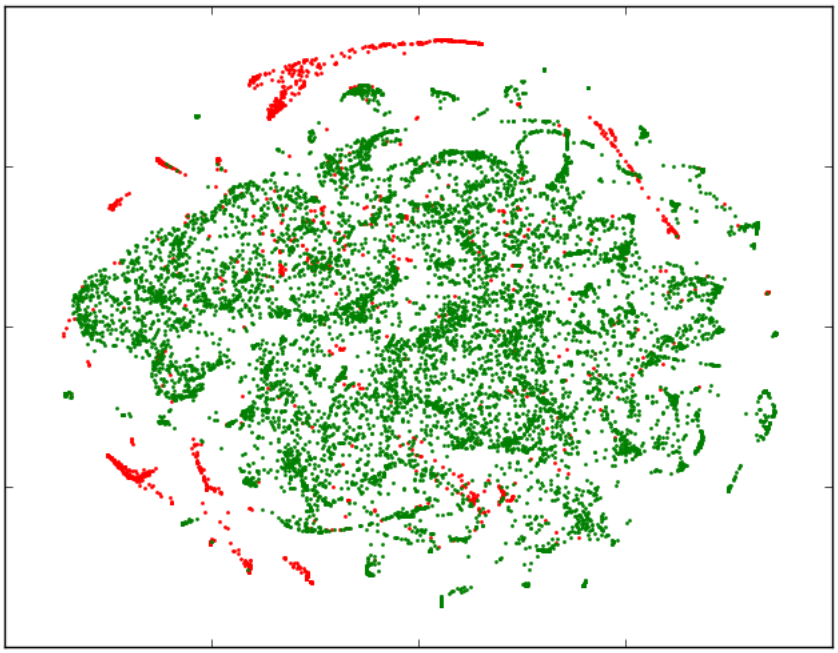


Fig. 6.1: Visualization of similarities of soft histogram representations of message sets in the *Large scale HTTPS* dataset by the t-SNE algorithm. Red dots represent message sets that were labeled as malicious while green dots represent benign message sets.

6.1 Similarity search on histograms

Before describing the actual method of the similarity search, we formally define the problem that is being solved.

We consider the following scenario: Given two sets, a reference set $\mathbb{S}$ (which contains annotated fingerprints) and a query set $\mathbb{Q}$ (which contains unknown fingerprints), of histogram fingerprints that are in a form of vectors from a d-dimensional space $\mathbb{R}^d$, the task is to find for each fingerprint $q \in \mathbb{Q}$ the k nearest fingerprints from the reference set $\mathbb{S}$ with respect to a given distance function $\delta : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$.

The k-NN query result set is then defined for $k \in \mathbb{N}$, $q \in \mathbb{Q}$ and the reference set $\mathbb{S}$ as:

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

The goal is therefore to perform k-NN queries for all objects in the query set $\mathbb{Q}$ and return their results. We consider batch processing of all queries from $\mathbb{Q}$. This operation is called *similarity join* of the sets $\mathbb{S}$ and $\mathbb{Q}$ in literature from the field of database research.

Results of the k-NN queries can be utilized in multiple ways — either they can be passed directly to an analyst for further analysis or information about their elements can be aggregated in order to directly provide a decision about the query object in which case the similarity search is equivalent to a k-NN classifier or detector.

6.2 Efficient k-NN search using metric indexing

To perform the k-NN search fast and effectively on large volume of data, a database index [110] is needed that provides either exact or approximate k-NN search over a large set of high-dimensional sparse fingerprints of the communication. 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 while preserving distances or the locality sensitive hashing [58] that tries to map close vectors to the same buckets with high probability.

Within the vast portfolio of database techniques implementing the two principles, we focus on metric indexing (specifically the M-Index [102]) which represents a metric variant of the locality sensitive hashing [103]. M-Index is a suitable basis for the task as it represents efficient yet extensible solution for fast similarity searches.

6.2.1 Basic principles of metric indexing

First, we formally introduce a metric space on top of which metric indexing operates.

A *metric space* $(\mathbb{U}, \delta)$ consists of a domain $\mathbb{U}$ (in our case, the domain is a space of all possible fingerprints, i.e. $\mathbb{U} \subseteq \mathbb{R}^d$) and a distance function δ which satisfies the metric axioms of *identity*, *non-negativity*, *symmetry*, and *triangle inequality*, defined $\forall x, y, z \in \mathbb{U}$ as:

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

Metric indexes ([19, 28, 24, 148, 23, 102]) organize database objects from the reference set $\mathbb{S}$ by grouping them based on their distances, with the aim of minimizing not only traditional database cost like input/output operations but also the number of distance function evaluations. The fundamental trick of metric indexes lies in using lower bounds that can be used to filter out irrelevant objects from the search cheaply and, thus, avoid unnecessary comparisons of objects.

To illustrate the lower bound principle, let us assume that in order to create the index structure, so called pivot objects $\mathbb{P} = \{p_i\}$ were selected from the set $\mathbb{S}$ and all distances $\delta(p_i, o), p_i \in \mathbb{P}, o \in \mathbb{S}$ were precomputed and are kept in the memory as a part of the index structure. Let us further assume that we are performing a k-NN search for a query object q for which the distances $\delta(p_i, q), p_i \in \mathbb{P}$ are known, too. Now, for a certain object $o \in \mathbb{S}$ and a pivot $p \in \mathbb{P}$, we would like to decide whether the object o lies within a query ball centered at q with radius r (i.e., to know whether $\delta(q, o) < r$) without explicitly computing the distance $\delta(q, o)$. Thanks to the triangle inequality, we can safely decide that the object o lies outside the query ball if the triangular lower bound $\delta_{\mathrm{T}}(q, o) = |\delta(p, q) - \delta(p, o)|$ is greater than the query ball radius r without need to know the value $\delta(q, o)$. This lower bounding principle is illustrated in Figure 6.2.

Therefore, an important feature of a metric index is that it keeps pivot–objects distances and uses them to filter out objects during the search queries.

6.2.2 Metric index for k-NN search on histograms

There have been developed many metric indexes varying in the application purpose, however, for purpose of the similarity search on histogram fingerprints, we introduce a simple main-memory variant of the state-of-the-art structure M-Index [102] that fits the requirements of

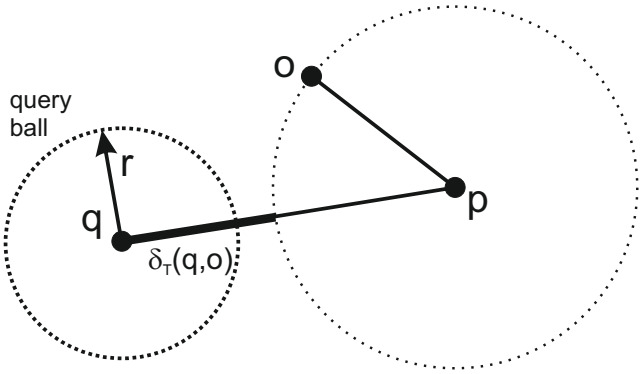


Fig. 6.2: Illustration of the lowerbounding principle.

the problem (high-dimensional but sparse data, relatively cheap distance computation due to their sparsity, many queries processed). Inspired by the approaches used in [19], [102] and [28], we assume a set of pivots $\mathbb{P} = \{p_i | i = 1, ..., n\}$ is selected from the reference set $\mathbb{S}$, each defining a partition in the given metric space. The remaining objects are partitioned such that each object is assigned to the partition of its nearest pivot. In such a way we obtain the Voronoi partitioning [5] (complete and disjoint), see Figure 6.3. The partitions are therefore further called Voronoi cells $\mathbb{C}_1, ..., \mathbb{C}_n$. Moreover, for each partition we store the distance r_i between its pivot and the furthest object within the partition (so called *partition radius*):

$$r_i = \delta(p_i, o), o \in \mathbb{C}_i, \forall o_j \in \mathbb{C}_i : \delta(p_i, o_j) \leq \delta(p_i, o).$$

Radius r_2 of the Voronoi cell $\mathbb{C}_2$ is marked by the dotted circle in Figure 6.3. Moreover, for each cell we also store the distances from the objects within that cell to its pivot p_i. Therefore, we keep the values

$$\delta(p_i, o_j), o_j \in \mathbb{C}_i$$

These distances are marked by the dashed lines in Figure 6.3 for the cell $\mathbb{C}_2$.

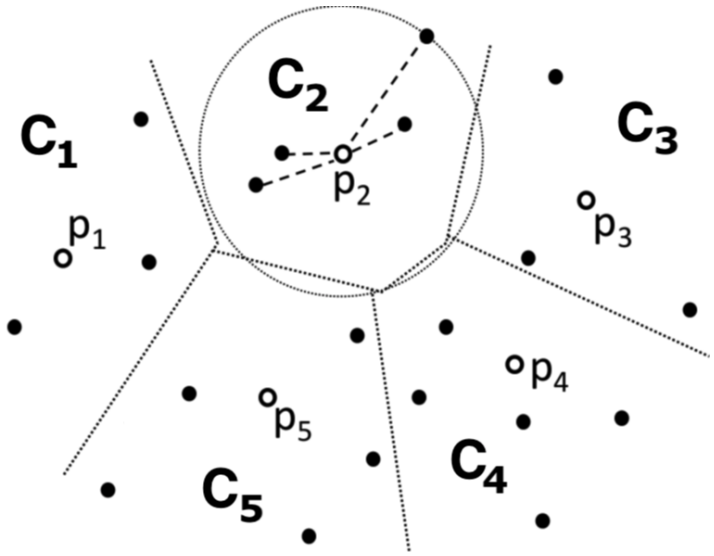


Fig. 6.3: Illustration of the Voronoi partitioning (here in 2D space) and the index built on top of it. The dotted lines mark boundaries between the Voronoi cells.

The resulting index corresponds to one level GNAT [19] or M-index [102]. The structure stores radius of each cell and distances from cell objects to corresponding pivots similarly as utilized by the M-Tree [28] (where they are called *distances to parent*). Note that also the M-Index considers the distances from cell objects to cells' pivots to construct a key value that is stored in the B-Tree [11] structure and used later for efficient searching.

In the following subsection, we describe how the index structure introduced here is utilized for optimization of k-NN search queries.

6.2.3 k-NN search optimization

Given a k-NN query for a query object $q \in \mathbb{Q}$ and $k \in \mathbb{N}$, the search progresses in the following way: First, Voronoi cell with the pivot which is the nearest to q is searched, then the cell with the second nearest pivot is searched, and so on. During the search, the nearest k objects to q found in the cells searched so far are maintained as the k-NN candidates. The search ends when all cells have been searched and the actual k-NN candidates are returned as the query result set. In order to speedup the search, the following optimizations that leverage the index structure are applied:

1. When a cell $\mathbb{C}_i$ is about to be searched, its radius r_i is checked whether the cell ball centered in p_i with radius r_i overlaps with the query ball or not. The query ball is formed by a ball centered in q with radius equal to the distance to actual k-th nearest neighbor candidate. If the query ball and the partition ball do not overlap, the entire cell $\mathbb{C}_i$ can skipped.
2. Whenever an object o from an actual cell 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 $\delta_{\mathrm{T}}(q, o)$ is computed (as described in Section 6.2.1). If the lower bound is greater than the actual query radius, the object can not become a k-NN candidate, hence the object is filtered without computing the distance $\delta(q, o)$.
3. It is not always necessary to perform the exact k-NN search that could lead to exhaustive search in many of the partitions. In order to speed-up the search even more, the k-NN search can be stopped after a predefined number of objects were inspected (e.g., 4% of all objects in the database). In this case, the exact results of the k-NN query can not be guaranteed. Therefore, this approach is called *approximate k-NN search* or *approximate similarity join*.

6.3 MapReduce implementation

In this section, we describe parallelized implementation of the similarity search on histogram fingerprints which was described in the previous section. The implementation uses the MapReduce programming model to achieve parallel processing of the data. We describe building of the histograms from input tuples for both the reference set $\mathbb{S}$ and the query set $\mathbb{Q}$ (Section 6.3.2) which is followed by the (approximate) similarity join of these sets in order to obtain the k-NN query results for all histograms from the query set (Section 6.3.3).

6.3.1 MapReduce programming model

Before we dive into details of the implementation, we present a brief overview of the MapReduce programming model which is used for the implementation.

The MapReduce model [35] was designed for massive parallelization to allow processing of large data on a cluster of computers which are called nodes. The cluster contains one *master* node which manages the computation and multiple *worker* nodes that realize the distributed computations. The master node accepts the request for a computation and distributes it over the worker nodes. The entire computation is composed of two phases — the *Map* phase and

the *Reduce* phase. In the Map phase, a user defined *map* function is applied on the input data that are stored in a form of *(key,value)* pairs and transforms each pair a to new *(key2,value2)* pair. When the *map* function is performed on all input pairs, the Map phase finishes and the the Reduce phase is started. In the Reduce phase, a user defined *reduce* function is applied which aggregates the *(key2,value2)* pairs that share the same key to produce the final result. Worker nodes that realize the Map phase are called *mappers*, while worker nodes that realize the Reduce phase are called *reducers*.

An important feature of the MapReduce model is that the *map* function is applied in parallel on all input pairs while the *reduce* function is applied in parallel on all groups that share the same key. Thanks to this, high degree of parallelization of the processing can be achieved. On the other hand, the user needs to take this into account when the data format and the *map* and *reduce* functions are designed in order to use the MapReduce model effectively.

6.3.2 Histograms building implementation

This MapReduce program performs transformation of raw input data — the tuples that contain the observed feature values — to soft histogram representations as described in Chapter 5. We assume that the input tuples are stored in a tabular format in the distributed files system of the MapReduce cluster and can be directly read by the mappers. Each row in the input table contains one tuple of observations and an identifier, which specifies the histogram to which the given tuple should contribute.

In the Map phase, $(key; value)$ pairs for further processing are generated. Each pair is generated from one row of the input table. The *key* part is composed of the identifier which specifies which histogram will be updated. The *value* part then contains the values from the given tuple (the observations) which are used to update the histogram.

The Reduce phase then aggregates the tuples with the same key into one histogram, using the soft histogram update procedure described in Chapter 5. As we can expect that most of histogram's bins will be empty (equal to zero), we use the sparse format for storing the created histograms. More specifically, each soft histogram is stored using sparse representation as a set of pairs $(binID : value)$, where pairs with value equal to zero are omitted. The reduce phase outputs the *key* which remains the same (in order to keep the identifiers of the produced histograms) and the created soft histogram. A scheme of the entire histograms building program is shown in Figure 6.4.

We note that building of histograms is a very suitable task for the MapReduce programming model because the individual updates of one histogram are independent. Hence, the order in which they are applied to build the histogram does not influence the result and the parallel processing can be effectively used.

6.3.3 Similarity search implementation

The second MapReduce program is dedicated to performing the similarity join of the sets $\mathbb{S}$ and $\mathbb{Q}$ that were created by the Histograms building program.

A MapReduce implementation of exact similarity join has been introduced in [90], where the authors propose a method employing the Voronoi partitioning and metric filtering rules for the index structure and replication of the Voronoi cells to all reducers (where the k-NN queries are computed) for parallelized exact similarity join processing. The replication of Voronoi cells among reducers is needed to ensure the exact results of k-NN searches, because in order to achieve parallel processing of the queries, the query set is distributed over the reducers and a k-NN query for an object $q \in \mathbb{Q}$ is computed using only the data that were assigned to the same reducer as the query object q.

Despite that the authors propose several filtering rules to avoid total replication of all Voronoi cells to all reducers, good efficiency results of exact similarity joins were reported only

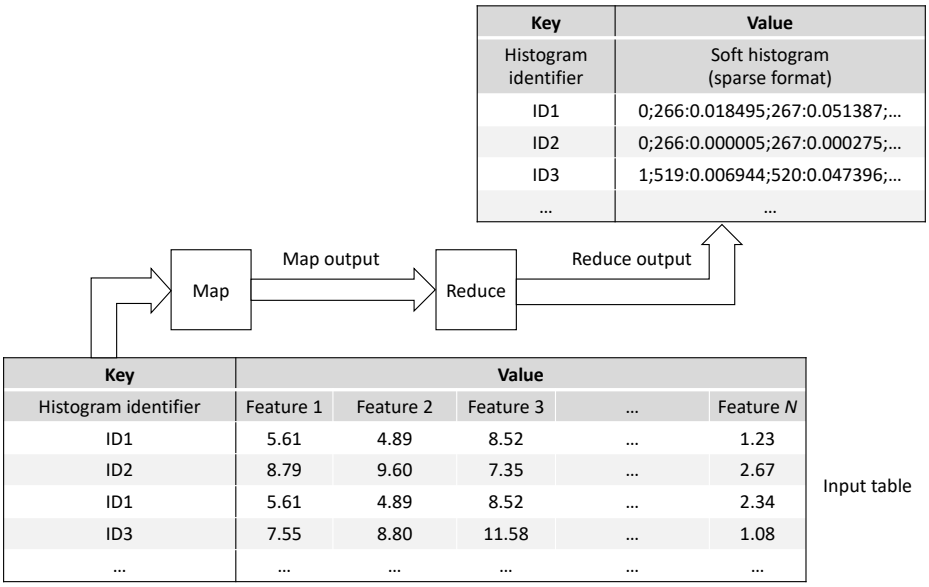


Fig. 6.4: A scheme of the histograms building program.

for data with different properties than of the soft histograms used here. Whereas in their paper [90] the authors used low-dimensional real vectors, the soft histograms investigated here are sparse but potentially high-dimensional vectors, with possibly imbalanced sparsity settings across the whole dataset. This prevents direct application of the method [90] in our scenario unless the MapReduce cluster would be really large and composed of worker nodes with high computational power. Therefore, we design here a method for the parallelized similarity join on soft histograms which uses approximate k-NN searches within each reducer in order to limit the replications of data and, thus, the computational requirements on each reducer.

The proposed method can be viewed as composed of three main steps:

1. *Preprocessing* — the index structure based on Voronoi partitioning is prepared and objects from the reference set $\mathbb{S}$ and query set $\mathbb{Q}$ are assigned to their respective Voronoi cells.
2. *Replication* — Voronoi cells are distributed to reducers. Furthermore, reference objects from each Voronoi cell are replicated to limited number of other reducers.
3. *Approximate similarity search* — k-NN similarity searches are performed on the reducers and the results are returned.

These steps are in more detail described below and also illustrated in Figure 6.5.

Preprocessing

In this step, pivots $\mathbb{P} = \{p_i | p_i \in \mathbb{S}, i = 1, ..., n\}$ from the reference set $\mathbb{S}$ are first selected. Since we assume large sets of data, random pivots selection is employed as a sufficient and not computationally demanding method. The exact number of pivots can be determined based on the volume of data and parameters of the MapReduce cluster. Given the set of selected pivots $\mathbb{P}$, the partitioning of objects from both sets $\mathbb{Q}$ and $\mathbb{S}$ to Voronoi cells $\mathbb{C}_1, ..., \mathbb{C}_n$ according to the selected pivots is performed (as described in Section 6.2.2).

When the partitioning is computed, the Voronoi cells are ready to be distributed to reducers. However, the number of pivots and, thus, the number of cells will be typically larger than the number reducers. Therefore, multiple cells need to be assigned to each reducer. To specify which cell will be assigned to which reducer, groups of cells $\mathcal{G}_1, ..., \mathcal{G}_N$ are created

(where N is equal to the number of reducers that will be used for running the similarity searches), where the group $\mathcal{G}_j$ contains Voronoi cells that will be assigned to the j-th reducer. In order to balance the workload of individual reducers, the cells are put into groups by using a modification of the geometrical grouping algorithm used in [90]. While the original algorithm builds the groups such that the total counts of objects assigned to individual reducers are balanced, we introduce an algorithm which balances the total sizes of objects on the reducers. The motivation for this approach is the observation that even though that the histogram fingerprints are generally sparse, the absolute number of non-zero bins can differ. Therefore, the grouping algorithm aims at balancing the smaller and larger histograms on each reducer. Pseudocode of this grouping algorithm is presented in the listing of Algorithm 1.

Algorithm 1 Geometric grouping by size

```
N = number of reducers
p_k = max_{p_j ∈ ℙ} Σ_{p_i ∈ ℙ} δ(p_i, p_j) //find the most distant pivot from others
P = ℙ\{p_k}  //all pivots except the initial one
U = {p_k}                //used pivots
G_1 = {ℂ_k}              //first group initialized
G_2, ..., G_N = ∅
for (i = 2; i ≤ N; i++) do //initialize the remaining groups
    p_k = max_{p_j ∈ P} Σ_{p_i ∈ U} δ(p_i, p_j)
    P = P\{p_k}; U = U ∪ {p_k}; G_i = {ℂ_k}
end
while P ≠ ∅ do
    G_i = group with the smallest total size of objects
    p_k = min_{p_j ∈ P} Σ_{p_i ∈ G_i} δ(p_i, p_j) //the nearest pivot to all pivots of cells in G_i
    P = P\{p_k}; G_i = G_i ∪ {ℂ_k}
end
return G_1, ..., G_N
```

Replication

After preprocessing, the groups of Voronoi cells are sent to reducers. Specifically, all the Voronoi cells from a group $\mathcal{G}_j$ are assigned to the j-th reducer. Therefore, after this assignment of cells to reducers, each reducer contains a subset of both the reference set and the query set. Additionally, replication of reference objects from each cell to selected reducers is performed. The scale of replication is determined by the *replication threshold* t_r which is used in the following rule:

Given a Voronoi cell $\mathbb{C}_i$, its pivot p_i and a reducer j ($\mathbb{C}_i \notin \mathcal{G}_j$), reference objects from the cell $\mathbb{C}_i$ are replicated to reducer j if:

$$\exists \mathbb{C}_k \in \mathcal{G}_j : \sum_{p' \in \mathbb{P}} [\delta(p_i, p') \leq \delta(p_i, p_k)] < t_r$$

Therefore, reference objects from $\mathbb{C}_i$ are replicated to the j-th reducer only if there is another cell on the j-th reducer, whose pivot is within the t_r nearest pivots to p_i.

Approximate similarity search

In this step, the k-NN search queries are performed on individual reducers. For a query object $q \in \mathbb{Q}$ assigned to the j-th reducer, the reference objects belonging to Voronoi cells from the group $\mathcal{G}_j$ and reference objects from cells that were replicated to the j-th reducer are searched. The search is performed in the similar way as described in Sectioin 6.2.3, including the optimizations in order to make the search more effective, namely the cells filtering rule

and the triangular lower bounds filtering. When the search finishes, the query result set for the given query object is sent to output. This procedure is performed for all query objects assigned to the given reducer. We note that the results are approximate, because only part of the original reference set $\mathbb{S}$ is searched on each reducer.

Implementation remarks

The three steps described above are in practice implemented as two MapReduce programs. The first program performs the preprocessing step, while the second program performs the replication step and the similarity search step. More specifically, the preprocessing is performed in Map phase of the first program (the Reduce phase is empty), while the replication step is performed in Map phase of the second program, followed by the similarity search step implemented in Reduce phase of the second program. For a pseudocode of the implementation, we refer to Appendix.

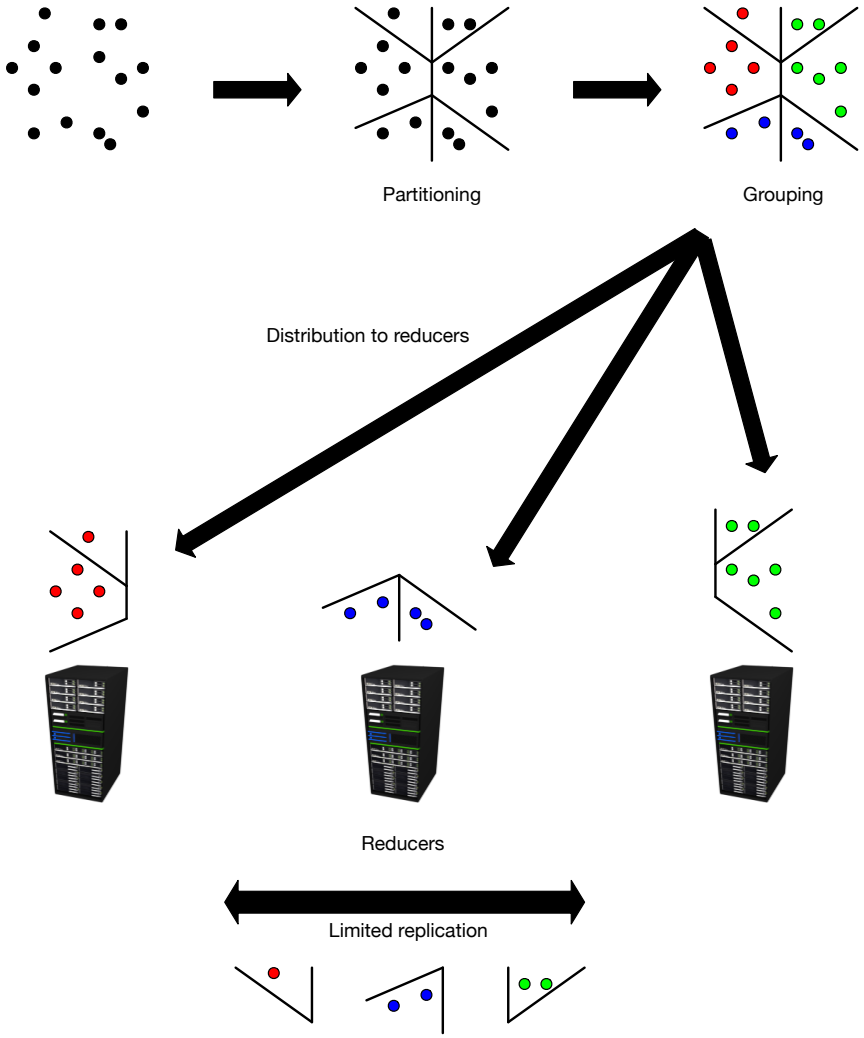


Fig. 6.5: Illustration of the MapReduce implementation of the approximate similarity search described in Section 6.3.3.

6.4 Experimental evaluation

In this section, we experimentally evaluate the proposed MapReduce solution for processing of histogram fingerprints. First, we demonstrate that the histogram fingerprints can be efficiently built using the MapReduce programming model. Then, we move to evaluation of the proposed approximate k-NN search method in a role malware detector. We show that even a small scale replication of reference objects is enough to significantly improve the detection performance which proves that the approximate similarity search is a promising way of improving scalability of a similarity search in the network security domain. Moreover, we show that a k-NN detector which uses the proposed method for approximate search outperforms a prior-art detector which was directly designed to optimize the FP-50 error rate [106] which is used to measure the performance of the detectors. Finally, we show that the proposed grouping strategy used to distribute data among the reducers is able to achieve more effective processing than the original method proposed in [90].

For all the evaluations below, we used the dataset *Large scale HTTPS* (Chapter 4, Section 4.3), because it is large enough to enable testing of the parallelized processing of message sets' representations on a cluster. Moreover, it presents a very challenging problem of malware detection when only logs of encrypted HTTPS communication are used. All the experiments presented here employ the soft histogram representations of message sets from the dataset. As the format of individual messages is similar to those used in experiments of Chapter 5 (they are 4-tuples of similar features), we used similar settings to build the soft histogram fingerprints of the message sets. Specifically, logarithmic transformation is applied on all features of the messages and we used equidistant bins centered at points $\{0, 1, ..., 10\}$ in each dimension. Therefore, the total number of bins in a joint soft histogram fingerprint was 11^4 similarly as in the previous chapter.

All the experiments were run on a Hadoop cluster with 100 worker nodes, each with 8GB RAM and 2 core CPU (Intel(R) Xeon(R) running at 2.20GHz).

6.4.1 Scalability of histograms building

The aim of the this experiment was to measure the gain of using the MapReduce framework for building the histogram fingerprints of message sets. We measured the computational time needed for building the fingerprints from the input data for different numbers of utilized mappers and reducers. All the 8 642 368 message sets composed of 145 822 799 messages were used for this experiment.

The process of building the fingerprints was described in Section 6.3.2. For the Map phase, the input data were split into blocks and each block was processed by one mapper. The number of reducers was set directly. The computational time needed for building the fingerprints depending on the number of mappers and reducers used is shown in Figure 6.6.

The graph demonstrates that moving the fingerprints building to the MapReduce environment is a promising and scalable way to process large volumes of data as even the relatively low number of reducers can significantly contribute to the decrease of time needed for the computation. Specifically, by using 80 mappers and 16 reducers, we were able to build representations of more than 8 million message sets in less than 90 seconds. While the number of mappers is not that important, even a low number of reducers significantly speeds up the computation. This is because the main part of the work is done on the reducers — the aggregations of message tuples into histograms, while the mappers only emit the tuples with the appropriate histogram identifier.

6.4.2 Classification

In this experiment, the proposed framework for approximate similarity search is evaluated in a role of k-NN classifier and compared to a reference prior art classifier. Moreover, we study

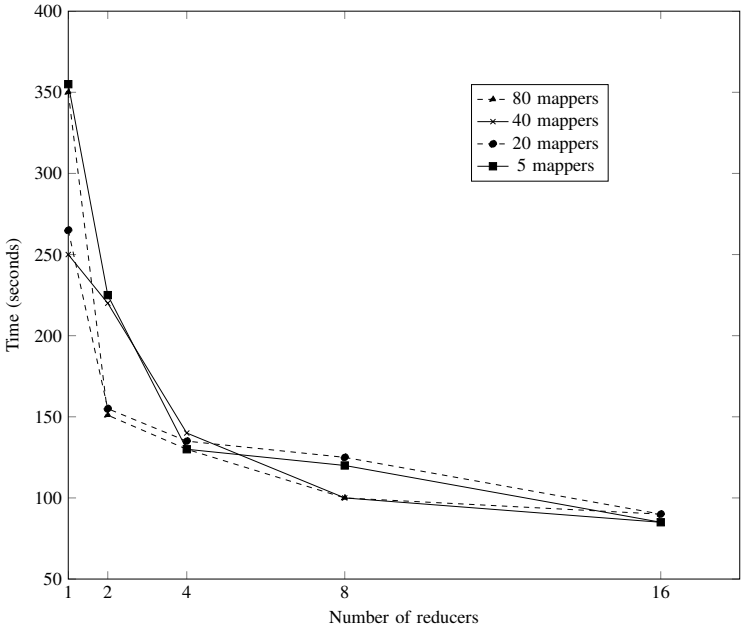


Fig. 6.6: Time needed for building the soft histogram fingerprints of message sets from the *Large scale HTTPS* dataset depending on the number of mappers and reducers used.

the dependence of the classifier's performance on the scale of replication of reference objects which is determined by the replication threshold t_r (see Section 6.3.3). We show that even a small replication is sufficient to significantly improve the classification results.

In order to evaluate the similarity search implementation as a classifier, the reference set $\mathbb{S}$ is treated as a training set for the classifier and the query set $\mathbb{Q}$ is treated as a testing set. Results in a query result set $kNN(q)$ for a query object $q \in \mathbb{Q}$ are aggregated using the inverse distance weighting to obtain continuous classification score as the classifier's output. The classification score is defined as:

$$s(q) = \frac{\sum_{x \in kNN(q)} \frac{1}{\delta(q,x)} [x \in \mathbb{S}^+]}{\sum_{x \in kNN(q)} \frac{1}{\delta(q,x)}},$$

where $\mathbb{S}^+ \subseteq \mathbb{S}$ is a set of positive objects (i.e., malicious message sets in this particular experiment) from the training set $\mathbb{S}$ and $[x \in \mathbb{S}^+]$ is the Iverson bracket notation, which has value 1, if the proposition inside the brackets is true, and 0 otherwise.

For evaluation of the classifier's performance, we use the FP-50 error measure proposed in [106] specifically for detectors in security domains which measures the false-positive rate at the level of 50% recall of an evaluated classifier. Besides introducing the FP-50 measure, the work [106] also proposes a classifier called Exponential Chebyshev Minimizer (ECM), which aims at minimizing the FP-50 error in its training process (for details, see Appendix). Therefore, we compare this reference ECM classifier to the k-NN classifier based on the similarity search framework proposed here.

Comparison of the k-NN classifier and the ECM classifier is summarized in Table 6.1. The table contains results averaged over 5 cycles, where in each cycle 1 000 000 message sets where randomly sampled from the dataset (out of 8 642 368) and used to create the testing set $\mathbb{Q}$ on which the classifiers were evaluated, while the remaining message sets (7 642 368) were used to form the training set $\mathbb{S}$. In each cycle, the parameter k of the k-NN classifier was selected from a set of possible values $\{5, 10, 15, 20, 25\}$ by cross-validation on the training set. The parameter t_r (replication threshold) was in each case fixed at the given value as shown

in Table 6.1. The k-NN search used Euclidean L_2 distance as the distance function δ and he number of pivots used for building the Voronoi partitioning was always fixed at 2000.

Classifier	FP-50 error (%)
k-NN ($t_r = 1$)	6.790
k-NN ($t_r = 2$)	0.739
k-NN ($t_r = 5$)	0.655
ECM	6.434

Table 6.1: Comparison of the k-NN and ECM classifiers, including three different values of the replication threshold t_r.

As we can see in Table 6.1, even for the very limited scale of replication ($t_r = 2$), the k-NN classifier is able to significantly outperform the ECM classifier which was designed to directly minimize the FP-50 error. On the other hand, the results show that at least minimal replication is needed to obtain good results ($t_r = 1$ actually means that no replication is performed at all).

Further study of the influence of replication on similarity search results is presented in Figures 6.7 and 6.8. The graphs confirm that even the small-scale replication significantly boosts the classifier's results, especially for higher values of k. Specifically, for $k = 20$ the classification error drops rapidly from more than 6% that was achieved without replication ($t_r = 1$) to less than 1% achieved with the minimal replication ($t_r = 2$) but then improves only slowly with further increases of the replication threshold. This confirms that only the nearest cells are important for the classification and that the proposed approximate similarity search performed only within the nearest cells of a query object is able to provide reliable results.

In case of small k, the influence of replication is not that significant because all the nearest neighbors are often found in the same Voronoi cell as the query object. Hence, the replicated cells do not influence the results at all and the higher FP-50 error is caused mainly by the insufficient number of nearest neighbors that are taken into account, not by the approximate similarity search.

6.4.3 Grouping strategy evaluation

This experiment was devoted to comparison of the grouping strategy of Voronoi cells proposed in Section 6.3.3 (Algorithm 1), which we further call *grouping by size*, to the original grouping strategy used by the state of the art implementation of the exact similarity search in [90], which we further call *grouping by count*. The experiment used similar settings as the previous one (Section 6.4.2) — namely the *Large scale HTTPS* dataset was used, number of pivots in Voronoi partitioning was set to 2000 and k was fixed at 20. Sizes of the sets $\mathbb{S}$ and $\mathbb{Q}$ were always 7 642 368 and 1 000 000 objects, respectively. Varied was the replication threshold and the grouping strategy.

Results of this comparison are shown in Table 6.2. For each combination of the replication threshold and the grouping strategy, we primarily measured the computational time needed to classify the set $\mathbb{Q}$ using the approximate similarity join of the sets $\mathbb{S}$ and $\mathbb{Q}$ and the FP-50 error achieved. Additionally, we measured the total number of reference objects in the database (which increases due to replication) and also the total number of pair-wise comparisons computed.

Foremost, the results prove that the proposed grouping by size strategy is able improve the computational time independently of the replication threshold used. Moreover, we can observe improvements in the FP-50 error as well. The computational time improvements are significant — from approximately 30% up to nearly 60% in case of $t_r = 2$. This improvement

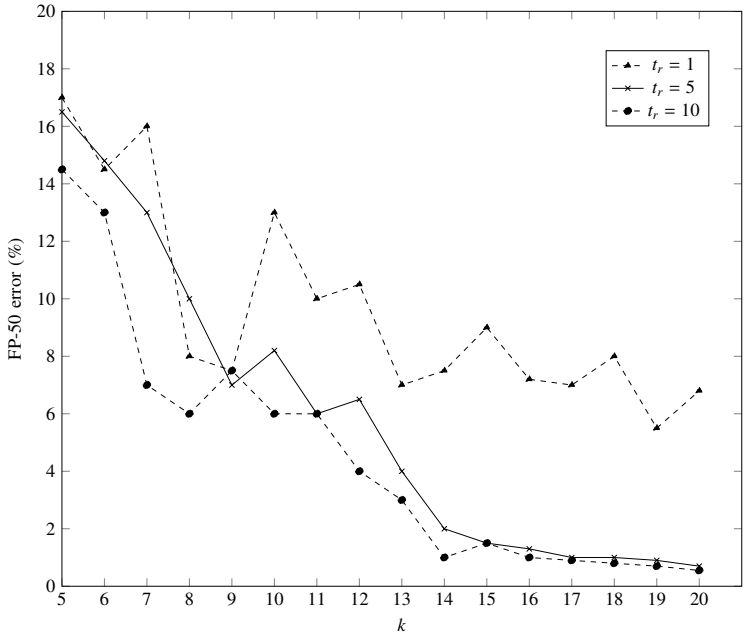


Fig. 6.7: FP-50 error for different values of k.

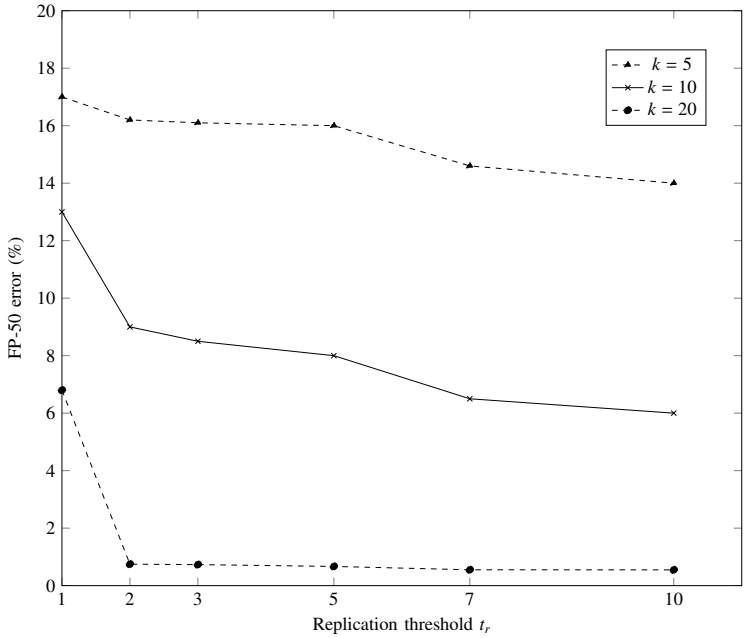


Fig. 6.8: FP-50 error for different values of the replication threshold t_r.

was achieved even though that the total number of reference objects and the total number of comparisons increased. This can be attributed to better distribution of the data among the reducers in case of the grouping by size strategy, which is exactly its goal. Moreover, in case of the grouping by count strategy, the computation was not able to finish at all for $t_r = 10$ (marked by dashes in Table 6.2).

Grouping	t_r	Time (s)	Ref. objects	Comparisons	FP-50 (%)
by size	1	1227	$7.6 \cdot 10^6$	$7.0 \cdot 10^9$	6.855
	2	1841	$11.1 \cdot 10^6$	$12.0 \cdot 10^9$	0.746
	3	2586	$14.1 \cdot 10^6$	$17.3 \cdot 10^9$	0.731
	5	4017	$18.7 \cdot 10^6$	$26.9 \cdot 10^9$	0.668
	7	7425	$23.1 \cdot 10^6$	$37.1 \cdot 10^9$	0.548
	10	14828	$28.9 \cdot 10^6$	$50.7 \cdot 10^9$	0.551
by count	1	2333	$7.6 \cdot 10^6$	$7.0 \cdot 10^9$	8.043
	2	4417	$11.0 \cdot 10^6$	$12.4 \cdot 10^9$	0.756
	3	5554	$14.0 \cdot 10^6$	$17.6 \cdot 10^9$	0.700
	5	8165	$18.4 \cdot 10^6$	$26.9 \cdot 10^9$	0.672
	7	10972	$22.5 \cdot 10^6$	$36.4 \cdot 10^9$	0.620
	10	–	–	–	–

Table 6.2: Comparison of the grouping strategies for different replication thresholds.

6.5 Chapter summary

This chapter proposed a framework for approximate similarity searches on large datasets of histogram fingerprints of network communication, enabling usage of the k-NN detector or classifier on such massive data. To achieve scalability, the framework is implemented using the MapReduce programming model which enables effective building of the histogram fingerprints and parallel processing of similarity search on top of them.

A method which uses approximate k-NN search was proposed to make the similarity searches on large datasets computationally feasible. The method uses parallelized similarity search processed on reducers in the MapReduce environment with limited replication of data among the reducers. The experimental evaluation proved that even a very limited replication of data is sufficient for significant improvement of the k-NN classification and outperforms a prior art classifier. Furthermore, improvement of the strategy used for distribution of data on the reducers was proposed which considerably decreases the computational time of the similarity search when compared to the strategy proposed in prior art.

A dataset created purely from logs of encrypted HTTPS traffic was used for evaluation of the classifier with promising results, showing that the k-NN similarity search is able to detect users infected with malware that communicates over HTTPS.

7

Representations based on kernel embedding of probability distributions

In previous chapters, we studied representations of message sets at different levels by means of joint sparse histograms. They proved to be effective for modeling communication with small number of features observed on the messages such that it is possible to construct the joint histogram. By their nature, the histograms fall into methods that explicitly estimate the probability density in order to enable comparison of the distributions. However, this explicit estimation is not always necessary. As we discussed in previous parts of this thesis, the main purpose of the representations is to enable reliable comparison of the distributions. To identify how similar or dissimilar are two message sets that capture the network behavior is the fundamental operation in most of the analytical tasks. Therefore, this chapter is devoted to designing a representation which would have the following properties:

1. There exists a distance metric on top of the representations, which can be used to measure distance between the underlying probability distributions that generated the represented message sets.
2. There is no need to explicitly estimate the probability density of the underlying distributions for purpose of measuring the distance, which is the case for many commonly used distance measures for probability distributions such as Kullback-Leibler divergence, Total Variation, Bhattacharyya distance, Hellinger distance or Rényi entropy [21] (we note that not all of these functions are indeed metrics as they do not satisfy all axioms of a metric), while preserving enough information about the distribution. This is important, because joint distributions can provide rich information about the features but direct estimation of the probability density would be often impossible due to demand for extremely high number of observations.
3. The representations can be stored as points from a $\mathbb{R}^d$ space such that they can be immediately consumed by most algorithms that assume input of this type.
4. Each representation can be easily updated with new observations as new messages are captured.

As we show below, solution for this can be found in the field of *kernel embedding of distributions* ([125, 97, 127]). Building on the results obtained for the feature maps in kernel methods for single data points, this approach extends the feature mapping to probability distributions by representing them as elements of a reproducing kernel Hilbert space. This then allows direct comparison of the distributions without estimating their density functions.

In this chapter, we therefore build on the idea of probability distributions embedding and study the applications of this approach in the domain of network communication representations.

7.1 Kernel functions and Hilbert spaces

Before we focus on embedding of probability distributions, we present a brief overview of basic properties of kernel functions and related Hilbert spaces upon which the entire field of study is built.

A *kernel* function k on a domain $\mathcal{X}$ is a symmetric function

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

Specifically, the most important in the area of machine learning are the positive definite kernels, which are kernel functions that fulfill the following condition (Mercer's condition):

$$\forall n \in \mathbb{N} \quad \forall x_1, ..., x_n \in \mathcal{X} \quad \forall c_1, ..., c_n \in \mathbb{R} : \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) \geq 0$$

Examples of positive definite kernels on $\mathcal{X} = \mathbb{R}^d$ include the linear kernel

$$k(x, y) = x^{\mathrm{T}} y$$

or the Gaussian kernel

$$k(x, y) = \exp(-\gamma \|x - y\|^2), \quad \gamma > 0.$$

Tightly connected with the positive definite kernels are the *reproducing kernel Hilbert spaces (RKHS)*. A reproducing kernel Hilbert space is a Hilbert space $\mathcal{H}$ of functions $f : \mathcal{X} \to \mathbb{R}$ with inner product $\langle f, g \rangle_{\mathcal{H}}$ for $f, g \in \mathcal{H}$ (and a norm $\|f\|_{\mathcal{H}}^2 = \langle f, f \rangle_{\mathcal{H}}$) which has the following reproducing property:

$$\forall x \in \mathcal{X} \quad \exists K_x \in \mathcal{H} \quad \text{such that} \quad \forall f \in \mathcal{H} : \quad f(x) = \langle f, K_x \rangle_{\mathcal{H}}$$

This means that if this property holds, a reproducing kernel k for the given Hilbert space $\mathcal{H}$ can be defined:

$$k(x, y) = \langle K_x, K_y \rangle_{\mathcal{H}}, \quad x, y \in \mathcal{X}$$

The connection between RKHS and positive definite kernels is that it can be proved that for each positive definite kernel k, there exists a Hilbert space $\mathcal{H}$ for which k has the reproducing property and vice versa — a kernel which has the reproducing property for some RKHS is positive definite [4].

The importance of RKHS and positive definite kernels for machine learning is that they enable the so called *kernel trick*. Given a domain $\mathcal{X}$ on which pair-wise comparisons of objects are made, e.g., for purposes of training a classifier, objects from $\mathcal{X}$ can be mapped to a Hilbert space $\mathcal{H}$ (referred as feature space) via a feature map $\phi : \mathcal{X} \to \mathcal{H}$ where the pair-wise comparisons of objects $x, y \in \mathcal{X}$ are replaced by the inner product $\langle \phi(x), \phi(y) \rangle_{\mathcal{H}}$.

Thanks to mapping the objects to the feature space which has higher dimension (possibly infinite), the analysis of data might become easier — for example, in classification tasks, two classes of objects might be separated by a classifier which is linear in the feature space which would not be possible in the original space $\mathcal{X}$. The problem is that computing the feature maps directly is often impossible (due to their infinite dimensionality). However, the properties of RKHS can be leveraged to compute the desired values of $\langle \phi(x), \phi(y) \rangle_{\mathcal{H}}$. without explicitly computing the feature maps.

If a positive definite kernel (which can be directly computed, e.g. the Gaussian kernel mentioned above) on $\mathcal{X}$ is used to express the pair-wise comparisons of objects from $\mathcal{X}$, then we know there exists a Hilbert space $\mathcal{H}$ for which the kernel k has the reproducing property $k(x, y) = \langle K_x, K_y \rangle_{\mathcal{H}}$. Therefore, we can choose the feature maps as

$$\phi(x) = K_x$$

which means. that we have

$$k(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}.$$

Thanks to this property, the comparisons in the feature spaces can be computed without need of direct computation of the feature maps which would be infeasible.

7.2 Kernel embedding of distributions

In the kernel embedding of probability distributions, each distribution P over a domain $\mathcal{X}$ is represented as a point in a reproducing kernel Hilbert space $\mathcal{H}$ of functions $f : \mathcal{X} \to \mathbb{R}$ using so called mean map μ:

$$\mu_P = \int_{x \in \mathcal{X}} k(x, \cdot) dP(x),$$

where k is the reproducing kernel for the space $\mathcal{H}$. In practice, usually a limited sample $X = \{x_1, ..., x_n\}$ of n observations from each distribution is available, hence the mean map is estimated from that sample in the following way:

$$\mu_P = \frac{1}{n} \sum_{i=1}^{n} k(x_i, \cdot) \tag{7.1}$$

Note that the mean map is a function of one variable, which is marked by the $\cdot$ symbol in the definitions above. Therefore, $\mu(t) = \frac{1}{n} \sum_{i=1}^{n} k(x_i, t), t \in \mathcal{X}$.

Having the distributions represented by the mean maps, kernel functions working on top of them can be introduced. These kernels can be then used by a classifier similarly as kernels working with single points are used. The work [96] proposes multiple different kernels for probability distributions, e.g. the linear kernel:

$$\kappa(P, Q) = \langle \mu_p, \mu_Q \rangle_{\mathcal{H}}, \tag{7.2}$$

where P, Q are two probability distributions. This kernel can be empirically estimated from the finite sample sets of observations as:

$$\kappa(P, Q) \approx \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} k(x_i, y_j),$$

where $x_i \sim P, y_j \sim Q$ are the collected observations from the distributions P and Q. Furthermore, similarly as Gaussian kernel or other non-linear kernels can be used instead of the linear one, the same can be done in case of probability distributions. The Gaussian kernel for distributions ([96]) is then defined as:

$$\kappa(P, Q) = \exp(-\gamma \|\mu_P - \mu_Q\|^2_{\mathcal{H}}) \tag{7.3}$$

The distances $\|\mu_P - \mu_Q\|^2_{\mathcal{H}}$ used in the Gaussian kernel (7.3) correspond to values of the *maximum mean discrepancy* (MMD) — a kernel two-sample test proposed in [61]. If the kernel k is characteristic ([61, 52], a kernel k is characteristic, if the mapping which maps a distribution P to its mean map μ_P is injective), then MMD defines a metric on the space of probability distributions and it holds that $\mathrm{MMD}(P, Q) = 0$ if and only if $P = Q$. This makes it suitable for use in various cases where similarities or distances between probability distributions are assessed. According to [61], MMD is defined as

$$\mathrm{MMD}(P, Q) = \sup_{f \in \mathcal{F}} \left\{ \mathrm{E}_{x \sim P}[f(x)] - \mathrm{E}_{y \sim Q}[f(y)] \right\},$$

where $\mathcal{F}$ is a unit ball in the reproducing kernel Hilbert space $\mathcal{H}$ ([4, 127]) associated with the kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ (here $\mathcal{X} = \mathbb{R}^d$).

In this chapter, we derive the representation of message sets from MMD, because it can be estimated from a small number of observations with enough accuracy, even if the observations are high-dimensional [61]. Moreover, it enables to compare joint distributions of features without explicitly estimating their probability density functions.

As already mentioned, an important advantage of MMD is that it can be well estimated from a limited set of observations. Given two sample sets of observations $X = \{x_i\}_{i=1}^{n_1}$,

$Y = \{y_i\}_{i=1}^{n_2}$ (with distributions P_X and P_Y), an estimate of $\mathrm{MMD}^2(P_X, P_Y)$ can be calculated ([61]) as

$$\mathrm{MMD}^2(X,Y) = \frac{1}{n_1^2}\sum_{i,j=1}^{n_1,n_1} k(x_i,x_j) + \\ + \frac{1}{n_2^2}\sum_{i,j=1}^{n_2,n_2} k(y_i,y_j) - \frac{2}{n_1 n_2}\sum_{i,j=1}^{n_1,n_2} k(x_i,y_j) \tag{7.4}$$

The above estimate can be compactly written as the norm of estimates of means in the Hilbert space $\mathcal{H}$ as

$$\mathrm{MMD}^2(X,Y) = ||\mu_X - \mu_Y||^2_{\mathcal{H}}, \tag{7.5}$$

where

$$\mu_X = \frac{1}{n_1}\sum_{i=1}^{n_1} k(x_i,\cdot), \qquad \mu_Y = \frac{1}{n_2}\sum_{i=1}^{n_2} k(y_i,\cdot) \tag{7.6}$$

are the empirical estimates of mean maps as in (7.1) of the distributions P_X and P_Y, respectively.

Although the MMD is a theoretically well justified distance on probability distributions, its practical use (not only in the field of network communication modeling) might be limited due to the excessive computational and memory requirements. Memory requirements grow linearly with size of the set of observations and system's memory could be quickly exhausted (when we do not want to discard old observations in order to avoid loss of information). Moreover, calculation of the distance between two sets of observations X and Y is of order $O(\max\{|X|,|Y|\}^2)$, which is prohibitive in practice. Therefore, the main limitations of direct computation of MMD are:

- Storing of possibly high number of observations which is increasing over time and makes it more difficult to use the sets of observations in algorithms employed for their further analysis based on their similarities or distances.
- Direct computation of full kernel matrices when two sets of observations are compared.

These constrains motivated our research for an alternative representation and approximate calculation that would replace the original MMD (7.5) with a calculation which has computational complexity independent of sizes of individual sets of observations.

In further text, we first review existing methods that aim at approximate computation of MMD in order to decrease its computational requirements. Then, we derive a new method for approximate MMD computation and observations storing which enables to represent the sets of observations as real vectors of fixed dimension that can be easily updated with new observations. The computation of full MMD is replaced by L_2 distance on the vectors that represent the sets of observations.

7.3 Related works on kernel and MMD approximations

This section provides overview of methods used to speed up computation of kernel matrices and kernel function evaluations. Then, we review methods that aim at approximation of computation of MMD as some of them build upon the methods developed for kernel approxiamtions.

7.3.1 Kernel function approximations

Nyström approximation

A well established way which proved to be useful in many machine learning tasks to speed-up computation of large kernel matrices takes use of methods based on low-rank approximations

of the full kernel matrix. Given a kernel $k(x, y)$ and n points $x_1, ..., x_n \in \mathbb{R}^d$, the methods are based on approximating the full kernel matrix $\mathbf{K} : \mathbf{K}_{ij} = k(x_i, x_j))$ in the form:

$$\mathbf{K} \approx \mathbf{C}\mathbf{W}^{-1}\mathbf{C}^{\mathrm{T}},$$

where $\mathbf{C}$ is a $n \times c$ matrix composed of c columns selected form $\mathbf{K}$ (i.e., $\mathbf{C}_{ij} = \mathbf{K}_{ij}, i = 1, ...n, j = 1, .., c, c < n$) and $\mathbf{W}$ is a $c \times c$ matrix defined as $\mathbf{W}_{ij} = \mathbf{K}_{ij}, i, j = 1, ..., c$. This approximation allows to find solution for a kernel-based classifier or predictor in time $O(c^2 n)$, which for c significantly lower than n means significant speed-up, because the time complexity without using any approximations might be $O(n^3)$ [6] (as the training procedure of a classifier often involves kernel matrix inversion or eigenvalue decomposition). Works in this area differ in the way how the c columns are selected. For example, [144] chooses the c columns from $\mathbf{K}$ uniformly at random without replacement. In [39], the columns are sampled with replacement (i.e., by c independent trials) and with respect to a non-uniform probability distribution over all columns of $\mathbf{K}$ which is dependent on the input data. Moreover, the matrix $\mathbf{W}$ is replaced by its rank-k approximation $\mathbf{W}_k$ (which further reduces the computational cost) and Moore-Penrose ([95, 15, 105]) generalized inversion $\mathbf{W}_k^+$ is used to ensure that it can be always computed.

In general, approximations of kernel matrices based on the Nyström method are suitable for cases when the $n \times n$ square kernel matrix is very large and the data are static in the sense that new observations are not continuously added to the dataset (e.g., for training a classifier). For purposes of MMD approximation, the Nyström method could be considered for computing the kernel matrices between two sets of observations. However, in the area of network communication modeling, we are mainly interested in cases when all the observations are not collected at once, but they are coming continuously as new messages are collected (e.g., from a long-term monitoring of network traffic and building models of behavior of individual users). In this scenario, it is crucial that the representations of individual sets can be simply and repeatedly recalculated as new observations arrive. For this reason, we do not consider approximations of this type for our work.

Random features

A Monte Carlo approximation of the kernel computation was proposed in [113]. The method builds on Bochner's theorem [57, 118] stating that for a continuous positive definite kernel $k(x, y), x, y \in \mathbb{R}^d$ (e.g., the Gaussian kernel), there is a probability measure p such that:

$$k(x, y) = \int_{\mathbb{R}^d} p(\omega) \exp(i\omega^{\mathrm{T}} \Delta) d\omega,$$

where i is an imaginary unit and $\Delta = x - y$[1]. Because here $\Delta \in \mathbb{R}^d$ and p is a probability measure over $\mathbb{R}^d$, the above integral can be rewritten as ([113]):

$$k(x, y) = \int_{\mathbb{R}^d} p(\omega) \cos(\omega^{\mathrm{T}} \Delta) d\omega,$$

In other words, the value of kernel k in the points x, y can be expressed as an expected value of function $\cos(\omega^{\mathrm{T}} \Delta)$ when ω is drawn from the distribution $p(w)$:

$$k(x, y) = \mathrm{E}[\cos(\omega^{\mathrm{T}} \Delta)], \omega \sim p(\omega) \tag{7.7}$$

The method of kernel approximation proposed in [113] uses Monte Carlo estimation of the expected value (7.7). It holds that:

$$\cos(\omega^{\mathrm{T}} \Delta) = \cos(\omega^{\mathrm{T}}(x - y)) = \cos(\omega^{\mathrm{T}} x) \cos(\omega^{\mathrm{T}} y) + \sin(\omega^{\mathrm{T}} x) \sin(\omega^{\mathrm{T}} y) \tag{7.8}$$

[1] Accordingly to notations used in the related literature, we use the kernel $k(x, y)$ and its signature $k(x - y)$ interchnageably.

By defining

$$\phi_\omega(t) = \begin{bmatrix} \cos(\omega^{\mathrm{T}} t) \\ \sin(\omega^{\mathrm{T}} t) \end{bmatrix},$$

sampling L samples $\omega_1, ..., \omega_L$ from $p(\omega)$, and using the identity (7.8), the expected value (7.7) is approximated by using as:

$$k(x, y) \approx \phi(x)^{\mathrm{T}} \phi(y),$$

where

$$\phi(t) = \frac{1}{\sqrt{L}} \begin{bmatrix} \phi_{\omega_1}(t) \\ . \\ . \\ . \\ \phi_{\omega_L}(t) \end{bmatrix}.$$

The work [26] is inspired by the above described approach and proposes a technique to optimize both the quality of the approximation and dimensionality of the resulting feature maps ϕ. It approximates the kernel value using the feature maps ϕ_ω defined as follows:

$$\phi_\omega(t) = \begin{bmatrix} \sqrt{\alpha_\omega} \cos(\omega^{\mathrm{T}} t) \\ \sqrt{\alpha_\omega} \sin(\omega^{\mathrm{T}} t) \end{bmatrix},$$

where $\omega \in \Omega$ are now selected from a predefined fixed set Ω (i.e., not sampled as in [113]) and $\alpha_\omega \geq 0$ are non-negative weights of individual features in the feature map. Dimensionality of the resulting feature map ϕ is then given as:

$$D = \sum_{\omega \in \Omega} [\alpha_\omega > 0] D_\omega.$$

D_ω represents dimensionality of ϕ_ω, i.e. $D_\omega = 2$ for $\omega > 0$ and $D_\omega = 1$ if $\omega = 0$ (because $\sin(\omega t) = 0$ for $\omega = 0$). The goal of the work is to optimize values of α_ω-s such that both D and discrpenacy between the values of the original kernel and the approximation are minimized. Evaluation of the kernel values for purposes of the optimization is performed on a preselected fixed set of points. Limitation of this method is the dimensionality of the input observations if the kernel k is not additive - in such cases, size of the set Ω increases very quickly (possibly exponentially with the number of input dimensions).

Related to these approaches are [138], [139] and [129] and various other works that, in general, replace non-linear kernels in classifiers (e.g., SVMs) by linear kernels on feature maps that have higher dimension but they are sparse and they are effective to compute.

Random features have been considered for approximation of kernel two-sample tests including MMD in [89], [131] and most importantly in [151], which is in more detail reviewed further.

7.3.2 Approximations of MMD

In this subsection, we review methods that aim at approximation of MMD computation.

Linear MMD

An approximation of MMD computation which can be computed in $O(n)$ time for two sets of n observations $X = \{x_i\}_{i=1}^n, Y = \{y_i\}_{i=1}^n$ was proposed in [61] and [62]. The speed-up is simply achieved by not evaluating all the pair-wise kernel values between each pair of observations in the samples. In literature, this approximation is known as *Linear MMD* (due to the linear time complexity) and is computed as:

$$\mathrm{MMD}^2(X, Y) \approx \frac{2}{n} \sum_{i=1}^{\frac{n}{2}} k(x_{2i-1}, x_{2i}) + k(y_{2i-1}, y_{2i}) - k(x_{2i-1}, y_{2i}) - k(x_{2i}, y_{2i-1}) \qquad (7.9)$$

B-test

In [147], a generalization of the Linear MMD approach is proposed which approximates the MMD by dividing the data in both sample sets into blocks of size B on which the full MMD is computed (hence the name *B-test*). Then, the MMD is approximated by averaging the results for individual blocks.

For sample sets $X = \{x_i\}_{i=1}^n, Y = \{y_i\}_{i=1}^n$ and B which determines the size of one block, MMD for the b-th block is computed as:

$$\eta_b = \frac{1}{B(B-1)} \sum_{\substack{i,j=(b-1)B+1 \\ i \neq j}}^{bB} k(x_i, x_j) + k(y_i, y_j) - k(x_i, y_j) - k(x_j, y_i),$$

for $b = 1, ..., \frac{n}{B}$. The final approximation is obtained as an average of η_b-s:

$$\mathrm{MMD}^2(X, Y) \approx \frac{B}{n} \sum_{b=1}^{\frac{n}{B}} \eta_b$$

Time complexity of this approximation is $O(Bn)$ as the MMD computation for one block takes $O(B^2)$ time and it is repeated on $\frac{n}{B}$ blocks. By varying the parameter B, approximations of different quality and computational requirements can be obtained — from the Linear MMD to full MMD for $B = n$.

FastMMD

Applications of random features (reviewed above in Section 7.3) for MMD approximation were shortly discussed in [131]. In [151], they are used to develop the FastMMD algorithm which stems from the original work [113]. It uses the similar Monte Carlo estimation as described for random features above to compute kernel matrices for two compared sets of observations. Additionally, the computational complexity is further decreased by applying Harmonic Addition Theorem [99]. Thanks to this seed-up, FastMMD can be computed in time $O(Ln)$ (where L is the number of ω-s sampled for the Monte Carlo estimation, n is the number of observations in the compared sets) while the original MMD's computational time is quadratic in the number of observations n. However, this means that the speed-up is significant only in cases when $L \ll n$ (the authors assume L to be in hundreds in their experiments in [151]).

Because FastMMD is based on random features as proposed in [113], the sets of observations can be represented and continuously updated with new observations in the following way: Given two sets $X = \{x_i\}_{i=1}^{n_1}, Y = \{y_i\}_{i=1}^{n_2}$, a shift-invariant kernel k (e.g., the Gaussian kernel) and the probability distribution $p(\omega)$ from its Fourier transformation as described above, L samples $\omega_1, ..., \omega_L$ are drawn from $p(\omega)$ and ϕ_ω-s are computed for each observation x_i and y_j from X and Y, respectively. This is done similarly as computing ϕ_ω-s and the feature maps ϕ for individual points in [113] as described above. Then, representations of the sets X, Y are computed:

$$\phi(X) = \frac{1}{n_1} \sum_{i=1}^{n_1} \phi(x_i)$$

$$\phi(Y) = \frac{1}{n_2} \sum_{i=1}^{n_2} \phi(y_i)$$

Approximation of MMD between X and Y is then computed as the distance between $\phi(X)$ and $\phi(Y)$:

$$\mathrm{MMD}^2(X, Y) \approx \|\phi(X) - \phi(Y)\|^2$$

Representation of a set is therefore computed as a mean of representations of individual observations in the set. This allows the representations to be updated on the fly. The representation is of fixed dimension L which is determined by the number of samples drawn from $p(\omega)$.

7.4 Proposed approximate computation of MMD

As we discussed above, computational complexity of the original MMD is often prohibitive because of the quadratic number of computations needed to evaluate it. It is $O(\max(n_1^2, n_2^2))$ where n_1 and n_2 are the number of observations in samples that are compared. This means that in case of straightforward implementation, three kernel matrices of sizes $n_1 \times n_1$, $n_1 \times n_2$ and $n_2 \times n_2$ have to be computed. Furthermore, we demand reasonably compact representations of the samples that would allow not only their comparison using MMD but also easy updating as new observations arrive and their effective storage in memory.

In order to satisfy these needs, we below derive the representations of sample sets such that they are formed by vectors of fixed dimension whose L_2 (Euclidean) distances approximate the values of MMD computed between the sample sets. Our approach is partially inspired by works [124] and [139]. The above mentioned work [139] replaces the computation of a kernel function which is non-linear in the input space by a kernel on feature maps of finite dimension which is linear in the feature space. Hence, the original kernel $k(x, y)$ is evaluated as:

$$k(x, y) \approx \langle \phi'(x), \phi'(y) \rangle_{\mathcal{H}'},$$

where $\phi'(x)$ is an approximation of the original feature map $\phi(x)$ of finite dimension. The approximated feature maps are designed to be elements of a subspace $\mathcal{H}' \subset \mathcal{H}$ (where $\mathcal{H}$ is the original RKHS associated with the kernel k) which is a span of feature maps $\phi(z_1), ..., \phi(z_n)$ for preselected representative points from the input space $z_1, ..., z_n$. Coordinates of the approximated features maps ϕ' are then coefficients in the linear combination of $\phi(z_1), ..., \phi(z_n)$ such that the norm $\|\phi(x) - \phi'(x)\|_{\mathcal{H}}$ is minimized.

The method which we propose approximates the mean maps (7.6) as a linear combination of a fixed base set of basis functions $\mathcal{K} = \{k(l_i, \cdot) | i \in \{1, \ldots, |\mathbb{L}|\}\}$, where $\mathbb{L} \subset \mathbb{R}^d$ is a set of points chosen from the same space in which the observations exist. For now, it is assumed that the set $\mathbb{L}$ is already given and its construction is deferred to the end of this section. Given a sample set of observations $X = \{x_i\}_{i=1}^n$, to express its respective mean map μ_X in $\mathcal{K}$, i.e. to find coefficients of basis function amounts α, the following optimization problem is solved:

$$\alpha = \operatorname{argmin}_\alpha \left\| \sum_{i=1}^{|\mathbb{L}|} \alpha_i k(l_i, \cdot) - \mu_X \right\|_{\mathcal{H}}^2 \tag{7.10}$$

$$= \operatorname{argmin}_\alpha \left\| \sum_{i=1}^{|\mathbb{L}|} \alpha_i k(l_i, \cdot) - \frac{1}{n} \sum_{i=1}^{n} k(x_i, \cdot) \right\|_{\mathcal{H}}^2, \tag{7.11}$$

Denoting the kernel matrix of points from $\mathbb{L}$ as $\mathbf{K}_{ij} = k(l_i, l_j), \mathbf{K} \in \mathbb{R}^{|\mathbb{L}|,|\mathbb{L}|}$ and kernel matrix of points from $\mathbb{L}$ and points from X as $\mathbf{Q}_{ij} = k(l_i, x_j), \mathbf{Q} \in \mathbb{R}^{|\mathbb{L}|,n}$, the solution of (7.11) is

$$\alpha = \frac{1}{n} \mathbf{K}^{-1} \mathbf{Q} \mathbf{1}_n. \tag{7.12}$$

where $\mathbf{1}_n$ is a vector of ones of size n. The solution vector α is therefore of fixed dimension $\alpha \in \mathbb{R}^{|\mathbb{L}|}$, which depends on the cardinality of the set $\mathbb{L}$.

Let us now consider two sets X and X with mean maps expressed in the base $\mathcal{K}$ as α_X and α_Y. Their MMD distance can be then approximated as:

$$\mathrm{MMD}^2(X, Y) \approx \left\| \sum_{i=1}^{|\mathbb{L}|} (\alpha_X)_i k(l_i, \cdot) - \sum_{i=1}^{|\mathbb{L}|} (\alpha_Y)_i k(l_i, \cdot) \right\|^2 \tag{7.13}$$

$$= \alpha_X^{\mathrm{T}} \mathbf{K} \alpha_X - 2 \alpha_X^{\mathrm{T}} \mathbf{K} \alpha_Y + \alpha_Y^{\mathrm{T}} \mathbf{K} \alpha_Y. \tag{7.14}$$

Since $\mathbf{K}$ is positive semi-definite (if k is Mercer's kernel), there exists a Cholesky decomposition such that

$$\mathbf{K} = \mathbf{L}\mathbf{L}^{\mathrm{T}},$$

and $\mathbf{L}$ is a triangular matrix. Using $\mathbf{L}$, the approximated MMD distance (7.14) can be compactly expressed as

$$\mathrm{MMD}^2(X, Y) \approx \left\|\mathbf{L}^{\mathrm{T}}\alpha_X - \mathbf{L}^{\mathrm{T}}\alpha_Y\right\|^2. \tag{7.15}$$

Substituting the calculation of α from (7.12) and using the property of Cholesky decomposition that

$$\mathbf{K}^{-1} = (\mathbf{L}\mathbf{L}^{\mathrm{T}})^{-1} = \mathbf{L}^{\mathrm{T}^{-1}}\mathbf{L}^{-1},$$

the approximate MMD becomes

$$\begin{aligned}
\mathrm{MMD}^2(X, Y) &\approx \left\|\mathbf{L}^{\mathrm{T}}(\alpha_X - \alpha_Y)\right\|^2 \\
&= \left\|\mathbf{L}^{\mathrm{T}}\mathbf{K}^{-1}\left(\frac{1}{n_1}\mathbf{Q}^1\mathbf{1}_{n_1} - \frac{1}{n_2}\mathbf{Q}^2\mathbf{1}_{n_2}\right)\right\|^2 \\
&= \left\|\mathbf{L}^{\mathrm{T}}\mathbf{L}^{\mathrm{T}^{-1}}\mathbf{L}^{-1}\left(\varphi(\mu_X) - \varphi(\mu_Y)\right)\right\|^2 \\
&= \left\|\mathbf{L}^{-1}\left(\varphi(\mu_X) - \varphi(\mu_Y)\right)\right\|^2.
\end{aligned}$$

where $\varphi : \mathcal{H} \to \mathbb{R}^{|\mathbb{L}|}$ is a functional evaluating a mean map $\mu = \frac{1}{n}\sum_{i=1}^{n} k(z_i, \cdot)$ of a sample $\{z_i\}_{i=1}^{n}$ at points from $\mathbb{L}$:

$$\varphi(\mu) = \begin{bmatrix} \mu(l_1) \\ . \\ . \\ . \\ \mu(l_{|\mathbb{L}|}) \end{bmatrix}$$

Note that it holds that

$$\begin{aligned}
\varphi(\mu) &= \frac{1}{n}\mathbf{Q}\mathbf{1}_n \\
&= \frac{1}{n}\begin{bmatrix} \sum_{i=1}^{n} k(z_i, l_1) \\ . \\ . \\ . \\ \sum_{i=1}^{n} k(z_i, l_{|\mathbb{L}|}) \end{bmatrix}.
\end{aligned}$$

Therefore, after pre-computing the inverse $\mathbf{L}^{-1}$ which is relatively easy as the matrix $\mathbf{L}$ is triangular, MMD distance $\|\mu_X - \mu_Y\|_{\mathcal{H}}^2$ is replaced by Euclidean distance $\|\mathbf{L}^{-1}(\varphi(\mu_X) - \varphi(\mu_Y))\|_2^2$.

In other words, the MMD distance between μ_X and μ_Y in $\mathcal{H}$ is replaced by generalized Mahalanobis distance (as introduced in [143] and [115]) between $\varphi(\mu_X)$ and $\varphi(\mu_Y)$ in $\mathbb{R}^{|\mathbb{L}|}$, determined by the matrix

$$\mathbf{L}^{-1^{\mathrm{T}}}\mathbf{L}^{-1} = \mathbf{L}^{\mathrm{T}^{-1}}\mathbf{L}^{-1} = \mathbf{K}^{-1},$$

where the Mahalanobis distance $\delta_{\mathbf{M}}$ determined by a matrix $\mathbf{M} \in \mathbb{R}^{d\times d}$ is for $x, y \in \mathbb{R}^d$ defined as:

$$\delta_{\mathbf{M}}(x, y) = (x - y)^{\mathrm{T}}\mathbf{M}(x - y)$$

A representation of a set of observations is therefore computed and stored using the function φ in a form of an $|\mathbb{L}|$-dimensional real vector, which can be updated on-line as new observations are collected, which removes the need to keep all observations in memory individually. Storing the representation as $\varphi(\mu)$ is preferred to $\mathbf{L}^{-1}\varphi(\mu)$ (which would save

computational resources) as significant portion of its components can be close to zero (or very small such they can be replaced by zero without loss of accuracy), enabling more memory efficient storage in sparse structures.

In further text, this proposed representation is called Approximated MMD representation — AMRep. Computing the AMRep representation from an input set of observations thus includes these steps:

1. Forming the set $\mathbb{L}$ with respect to the given input set of messages.
2. Computing the matrix $\mathbf{L}^{-1}$.
3. Computing and maintaining the function φ for each message set.

7.4.1 Construction of the set $\mathbb{L}$

To this end it has been assumed that the set $\mathbb{L}$ is given. The method how $\mathbb{L}$ is constructed should reflect the intention of its use. For determining the set $\mathbb{L}$, we can expect there will be an input training set of observations $\mathbb{O}$ available, that can be used to compute $\mathbb{L}$. In a typical case, $\mathbb{O}$ will be a set of all observations obtained as training data for a classifier, or all observations contained in some unknown data that are about to be analyzed. A common requirement would be to select $\mathbb{L}$ such that $|\mathbb{L}| \ll |\mathbb{O}|$. Otherwise, storing of the resulting representations and all computations would be expensive.

If the representations will be used to model *majority* behavior, then items of $\mathbb{L}$ should be located in parts of the space where most observations are concentrated (i.e., in the most "populated" areas of the input space) and algorithms like self-organising maps [84] are appropriate. Contrary to that, if representations will be used to distinguish many different events or to detect rare events like in anomaly detection, elements of $\mathbb{L}$ should *cover* the entire input space, including regions with low density of observations. The latter approach is followed in the ongoing experiments, as it matches the needs in analysis of network communication which is the focus of this thesis. Formally, the cover set $\mathbb{L}$ should in this case satisfy the following condition:

$$\forall o \in \mathbb{O}\ \exists l \in \mathbb{L} : \|l - o\|_2 \leq \epsilon,$$

where ϵ is a preset constant. This parameter determines granularity (and thus sensitivity of the representation) and size of the cover $\mathbb{L}$. In experiments, we used this condition to build $\mathbb{L}$, when a sub-optimal solution to the class cover problem ([20, 91, 111]) was obtained by a greedy algorithm shown in the listing of Algorithm 2. However, the AMRep representation is not limited to this specific algorithm and any method which is able to construct $\mathbb{L}$ with desired properties can be potentially considered.

Algorithm 2 Greedy ϵ-cover of a set $\mathbb{O}$

```
1: procedure COVER(𝕆, ε)
2:     𝕃 ← {o}, o randomly selected from 𝕆
3:     for o ∈ 𝕆 do
4:         if ∀l ∈ 𝕃 : ‖o − l‖₂ > ε then
5:             𝕃 ← 𝕃 ∪ {o}
6:         end if
7:     end for
8:     return 𝕃
9: end procedure
```

7.5 Experimental evaluation — Classification

This section is devoted to experimental comparison of the proposed representation framework AMRep to the most relevant related methods [71], [72] and [42] (reviewed in detail in the

next subsection) from prior art. These works were selected because (i) they can be extended to different definitions of messages and message sets and (ii) they cover principles of most approaches proposed by the state of the art algorithms that rely on modeling of probability distributions.

Additionally, the soft histogram representations described in Chapter 5 and representations based on FastMMD (described above in Section 7.3.2) are included as well to compare pros and cons of both representations. Thanks to its design, FastMMD allows to build representations that satisfy our requirements that we set in the introduction of this chapter — it allows computing distances between probability distributions without estimation of probability densities and it provides representations that are of fixed dimension and can be updated on the fly. Moreover, as the authors show in their paper [151], FastMMD is able to achieve better approximation than the previously proposed approximations of MMD (Linear MMD and B-test). For these reasons, we include it in the experimental comparison as the most related method for approximate computation of MMD.

All the comparisons are made on the *TCP Flows* dataset (described in Chapter 4, Section 4.4) using both levels of communication which this dataset enables to use (the level of TCP packets referred as Any-P dataset and the level of TCP flows referred as Any-F dataset). The older datasets used in Chapter 5 could not be considered, because due to data retention policies, they were not available to us at the time of running these experiments.

7.5.1 Compared prior art methods

This subsection provides overview of the prior art methods that are used for the experimental comparison.

Multinomial Naive Bayes (MNB)

Hermann et al. [71] models encrypted communication by one-dimensional histogram of packet sizes with packets' direction encoded by sign. In the formal model introduced in Chapter 3, one message in [71] corresponds to one packet and a message set corresponds to a set of packets interchanged during loading of one specific web page. As individual messages are represented by scalar values (which are their sizes), MNB uses a one-dimensional hard histogram $h^{\mathcal{R}}$ to represent a message set $\mathcal{R}$. The histogram has m bins $h_1^{\mathcal{R}}, ..., h_m^{\mathcal{R}}$ and the i-th bin captures the number of messages of size i observed in $\mathcal{R}$. Each target class ω is represented by a set of empirical probabilities $\{P(i|\omega)|i = 1, ..., m\}$, where $P(i|\omega)$ is an estimation of probability that a message of size i will be observed in a message set belonging to class ω. An unknown message set $\mathcal{R}$ is then assigned to the most probable class ω^*, which satisfies:

$$\omega^* = \arg\min_{\omega} - \sum_{i=1}^{m} h_i^{\mathcal{R}} \log P(i|\omega).$$

Since the motivation behind this representation was to treat histogram bins as words, the histogram bins are scaled using term-frequency inverse document frequency (TF-IDF) paradigm. The TF-IDF scaling is applied on each bin of the histogram from which the probabilities $P(i|\omega)$ are estimated. The best results were reported for the combination of term-frequency normalization (which is implemented by using logarithmic scale for bins' sizes in the histograms) combined with L_2 normalizations of the histograms. These transformations were implemented in the below comparison. Therefore, this approach also represents usage of common empirical histograms with hard bins for representing the traffic.

Despite the above representation was originally proposed for fingerprinting web sites, it can obviously be used with other definitions of messages and message sets for classification of network communication.

Statistical Protocol Identification (SPID)

SPID [72] is designed to identify application protocols on basis of variable number of messages' attributes. Looking at this framework by the formalism used in this thesis, SPID represents a single message set as well as a model of a single protocol by a set of d empirical estimates of marginal probability densities of d features observed on the messages. A classified message set $\mathcal{R}$ is then assigned to protocol ω^* which minimizes the following Kullback-Leibler divergence averaged over the observed features:

$$\omega^* = \arg\min_{\omega} \frac{1}{d} \sum_{i=1}^{d} D_{\mathrm{KL}}(P_i^{\omega} \| P_i^{\mathcal{R}}),$$

where P_i^{ω} and $P_i^{\mathcal{R}}$ are estimates of probability density functions of the i-th feature for the protocol model ω and for the classified message set $\mathcal{R}$, respectively. Individual features are therefore considered to be independent of each other and only their marginal distributions are modeled, using conventional empirical hard histograms.

Tunnel Hunter (TunHunter)

Tunnel Hunter [42] identifies application-layer protocols tunneled through other protocols. Its main difference to all related methods is that it captures packets' order by modelling separately the first, the second, ..., up to the N-th packet in a TCP flow. Models of protocols are implemented as joint empirical hard histograms of packets' sizes and inter-arrival times and they are estimated from a set of TCP flows with the same protocol label. Unlike most of other works, TunHunter has stricter requirements for deployment, as the order of packets matters. The histograms are then smoothed by a Parzen window estimator (in implementation used for comparison in this work, Gaussian kernel was used for this smoothing as suggested in the original work). The classification is a variation of the likelihood-based classifier, where a message set $\mathcal{R} = \{m_i\}_{i=1}^{n'}$ (a TCP flow) is assigned to a protocol ω^* which minimizes the following log-likelihood:

$$\omega^* = \arg\min_{\omega} -\frac{1}{\min\{n', n\}} \sum_{i=1}^{\min\{n', n\}} \log P(m_i|\omega),$$

where n is the number of messages (e.g., packets) used to model each protocol, n' is the number of messages observed in the classified message set $\mathcal{R}$ and $P(m_i|\omega)$ is a probability of observing the i-th message $m_i \in \mathcal{R}$ (we note that in this case the message set is ordered) in a TCP flow belonging to the protocol ω.

7.5.2 Application identification

In this section, the AMRep representation of message sets is compared to the methods described in Section 7.5.1 on a problem of identifying an application to which the observed network communication belongs.

The comparison is done at the level of individual TCP flows, where one message corresponds to a single TCP packet (Any-P dataset) and at the level where a message set is identified by unique tuple of client IP, server IP and server's destination port, and one message corresponds to one TCP flow (Any-F dataset). Since AMRep representation is designed as a replacement of the original MMD distance between two probability distributions, it is primarily used with the k-nearest neighbors (k-NN) classifier [51] for which a meaningful definition of distance between two points is crucial. Also, the k-NN classifier naturally well extends to multi-class problems (recall that the dataset used in this section contains 69 different classes — the individual applications). Similarly, FastMMD and soft histogram

representations were used with the k-NN classifier too in order to rule out differences caused by different classification algorithm.

The implementation of the k-NN classifier was taken from the PRTools toolbox for Matlab [40], which has advantage in having its own routine for determining the proper value of k from the training data (using cross-validation). Other parameters of the AMRep representation (ϵ for the cover computation and γ for the width of the Gaussian kernel used in the mean map) were determined by exhaustive search over all combinations of values $\epsilon \in \{0.5, 0.6, ..., 2\}$ (Any-P dataset) and $\epsilon \in \{0.5, 0.6, ..., 5\}$ (Any-F dataset) and $\gamma \in \{2^n | n \in \{-2, -1, ..., 4\}\}$ for the kernel width values. Each combination of the parameters' values was evaluated by 5-fold cross-validation and the combination with the highest classification accuracy was used to train the final classifier which was then evaluated on a testing set. We note that the classification accuracy is defined as the ratio of the number of correctly classified messages sets to the total number of classified messages sets.

The evaluated soft histogram representation used 11 bins in each dimension similarly as in experiments in Chapter 5. Therefore, total number of bins in a histogram was 11^n where n was the number of features observed on the messages.

For the compared representation based on FastMMD, the parameters were also determined by cross-validation. Specifically, the width of the Gaussian kernel γ, which has the similar role as in the case of AMRep, was selected from the set $\{2^n | n \in \{-2, -1, ..., 4\}\}$ (i.e., similarly as for AMRep). The parameter L, which determines the size of the representation, was chosen (based on the results presented in [151]) from the set $\{300, 350, ..., 1200\}$.

Presented experimental results are averages over ten repetitions, where each iteration used 10 000 randomly selected samples (message sets) from the entire dataset (Any-P or Any-F). This sample was used to create the training and the testing sets in the given iteration. Each iteration used 10-fold cross-validation to estimate the classification accuracy. However, since ground-truth (true labels) is notoriously difficult to obtain in the network security, this scenario was simulated by using only 10% of samples available in each iteration for training in one fold of the cross-validation. Remaining 90% of samples were used to estimate the accuracy of identifying the application corresponding to testing samples. Average accuracy computed over all 69 applications (and all 10 iterations) is shown in Table 7.1, presenting results also for different combinations of messages's features used. According to the results, AMRep with k-NN outperforms representations and methods proposed in the prior works. Moreover, the results show that statistics of packets are more informative for application identification than statistics of flows, which is expected. Interestingly, according to these experiments, inter-arrival times of packets improved the accuracy only by 0.3% on the Any-P dataset. However, paired t-test at 5% significance level does not reject the hypothesis that the accuracies achieved with and without the inter-arrival times are the same. Hence, this improvement can be considered statistically significant.

Additionally, we will separately discuss differences in results of the representations AMRep, FastMMD and the representation based on soft histograms (SoftHist). As can be seen from the results in Table 7.1, AMRep outperforms both of the other representations. While the difference between AMRep and FastMMD is smaller, it is much higher with respect to the SoftHist representation. This shows that the kernel embedding of distributions can indeed capture much richer information than the histogram-based representations.

However, there are also differences in the size of the produced representations. This is compared in Table 7.2. From the table we can see that while FastMMD is able to achieve accuracy which is in certain cases comparable to AMRep, it is at the cost of significantly lower sparsity of the produced vectors (by sparsity we mean the number of non-zero elements of a vector). Technically, both representations (AMRep and FastMMD) produce dense representations, however some elements can be very close to zero (e.g., below 10^{-9}) and can be safely set to 0 in order to make the representation sparse. Therefore, for evaluation of sparsity of the produced representations, we counted as non-zero only those elements that were greater than 10^{-9}.

On the other hand, in case of soft histograms, the situation is opposite. The SoftHist's accuracy is lower than of the kernel-based representations (although still higher than of the methods from prior art) but it can benefit from extremely sparse representations (see Table 7.2) which allows to store the vectors very efficiently. Based on these results, we can conclude that on a smaller volume of data or if the computational requirements are not crucial, it is more desirable to use the representation based on kernel embedding of distributions because of the higher accuracy and richness of the representation. However, if a very large volume of data is about to be analyzed (e.g., as presented in Chapter 6), it might be beneficial to choose the soft histogram representation instead because of its effective processing.

Besides the accuracy, we also empirically estimated running times of each method in the classification phase. Runtimes (in seconds) of individual methods needed for computing classifications of the testing data in one fold (approximately 9000 message sets) are compared in Table 7.3. The table presents runtimes averaged over all folds of the cross-validation on a personal computer with Intel Core-i7 CPU (4 cores, 2.2GHz) and 16GB RAM. The evaluation environment was implemented in Matlab R2016a software.

First, we can conclude that the representations based on kernel embedding of distributions are able to achieve significantly higher accuracy without considerable time trade-off. Second, the computation times for soft histograms again confirm their effectivity as they are by more than 10% faster than the representations that use kernel embedding of distributions. Finally, the fast computation of MNB can be attributed to its simplicity, as it uses only one feature (message sizes) and a Naive Bayes classifier, which can be computed very quickly. However, it is at the cost of significantly decreased accuracy.

In order to provide better illustration of differences in the runtimes, relative comparison is provided in Figure 7.1 which shows the times relatively to AMRep (used with the k-NN classifier).

	Features	MNB	SPID	TunHunter	AMRep	FastMMD	SoftHist
Any-P	sizes	33.43%	41.23%	—	**66.83%**	62.93%	60.02%
	sizes, inter-arrival times	—	43.59%	58.31%	**67.19%**	63.03%	62.28%
Any-F	sizes	21.80%	39.67%	—	**51.94%**	46.75%	40.8%
	sizes, durations	—	43.21%	—	**53.29%**	48.05%	45.17%
	sizes, durations, packets	—	44.17%	—	**53.26%**	49.68%	49.03%

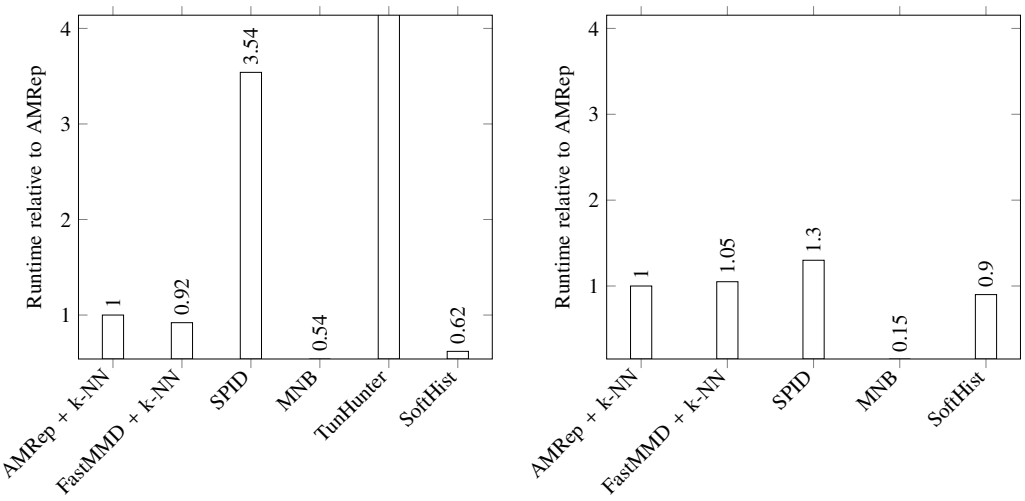
Table 7.1: Average accuracy of application identification on the Any-P dataset (top half of the table) and on Any-F dataset (lower).

	AMRep	FastMMD	SoftHist
Any-P (sizes)	11.2	272.2 (24.3)	4.7 (0.42)
Any-P (sizes, inter-arrival times)	171.8	579.1 (3.37)	13.2 (0.07)
Any-F (sizes)	18.7	495.3 (26.4)	6.8 (0.36)
Any-F (sizes, durations)	135.9	592.1 (4.36)	17.6 (0.13)
Any-F (sizes, durations, packets)	347.8	788.5 (2.27)	81.2 (0.23)

Table 7.2: Average number of non-zero elements (greater than 10^{-9}) of AMRep, FastMMD and SoftHist representations for different datasets. The value in parenthesis is the number of non-zero elements relative to AMRep.

Method	Any-P dataset	Any-F dataset
AMRep + k-NN	0.13s	0.20s
FastMMD + k-NN	0.12s	0.21s
SPID	0.46s	0.26s
MNB	0.07s	0.03s
TunHunter	4.36s	—
SoftHist	0.08s	0.18s

Table 7.3: Average execution times of individual methods needed for classification of the test data.



(a) Any-P dataset

(b) Any-F dataset

Fig. 7.1: Average execution times of individual methods, relative values to AMRep + k-NN classifier.

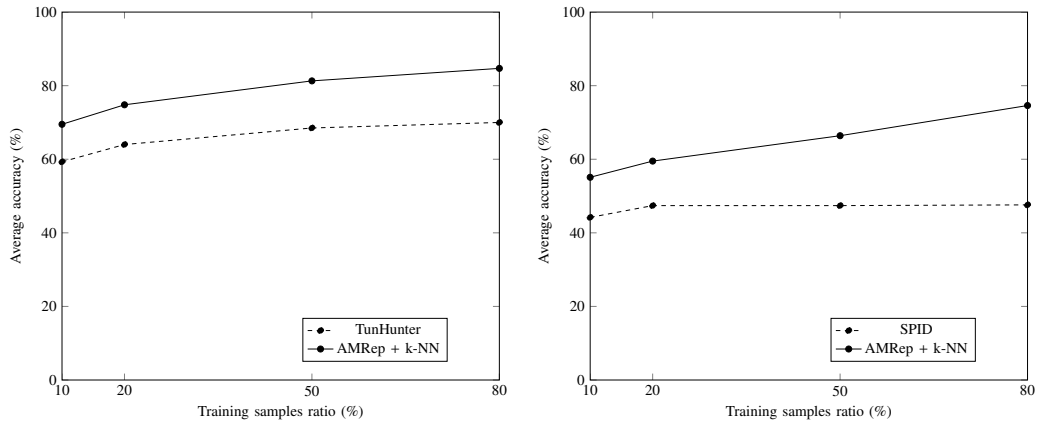
7.5.3 Further study of the AMRep representation

In this subsection, we further study properties of the AMRep representation and compare it to the prior art methods for traffic classification.

Training data influence

Since results in Table 7.1 used only 10% of available samples for training (see Section 7.5.2), it is interesting to study how the accuracy improves as the number of training samples increases. This has been investigated by holding 20% of samples in each iteration for testing and varying the size of the training set from 10% to 80% randomly selected from remaining 80% of samples. Resulting graphs for AMRep and the best prior art method (based on the results achieved in Section 7.5.2) for each version of the dataset are shown in Figure 7.2. As expected, the accuracy improves as the size of the training set increases, especially for models based on AMRep. Contrary, the accuracy of SPID quickly reaches flat region when the size of the training set is 20% of all data. SPID's behavior can be explained by histograms in its model being estimated with sufficient accuracy, therefore the approach hits its boundary and only

adding new features could probably further improve the accuracy. As seen in Figure 7.2, the accuracy of AMRep with k-NN classifier improves as the amount of training data increases, reaching accuracy of 84.7% on Any-P dataset and 74.6% on Any-F dataset when 80% of messages sets are used for training. This makes it always the best performing classifier on given dataset.



(a) Any-P dataset (b) Any-F dataset

Fig. 7.2: Comparison of average accuracy for AMRep and the best prior art methods depending on the ratio of samples used for training.

Confusion matrix

To further discuss AMRep's performance, Figure 7.3 shows confusion matrix in form of a heat map for all 69 applications on the Any-P dataset. Each row and each column in the figure corresponds to one application and the color of each cell (i, j) represents the ratio of message sets of application i classified as application j. The lighter the color of a cell, the higher the ratio which the cell represents. Cells on the main diagonal therefore represent accuracies achieved on individual applications while cells outside the diagonal represent misclassifications. Due to the large number of classes, the names of individual applications are omitted in the figure. The confusion matrix shows that for the vast majority of applications the classification is very accurate, which also means that statistics of their packets are very distinctive. However, there are few pairs which are frequently confused, some of them are discussed in detail below:

- `GoogleSoftwareUpdateAgent` and `ksfetch` — `ksfetch` is an OSX process involved in downloading updates for Google products, hence these two misclassified applications are related and might share the code base.
- `SoftwareUpdateCheck` and `softwareupdate` — Both of these are names of processes running on OSX systems and involved in checking for updates of software installed on the system. Hence, there is clear relation between these processes.
- `WmiPrvSE.exe` and `lsass.exe` These are processes that are essential parts of Windows operating systems. Their roles in the system are different but it might be possible that they share the same libraries for network communication. However, this would have to be verified by deep analysis of the implementations.
- `cma` and `DropboxOriginal` — `cma` is a part of McAfee software, while `DropboxOriginal` is part of the Dropbox file storage service. These two are likely unrelated and this is an example of a mis-classification.

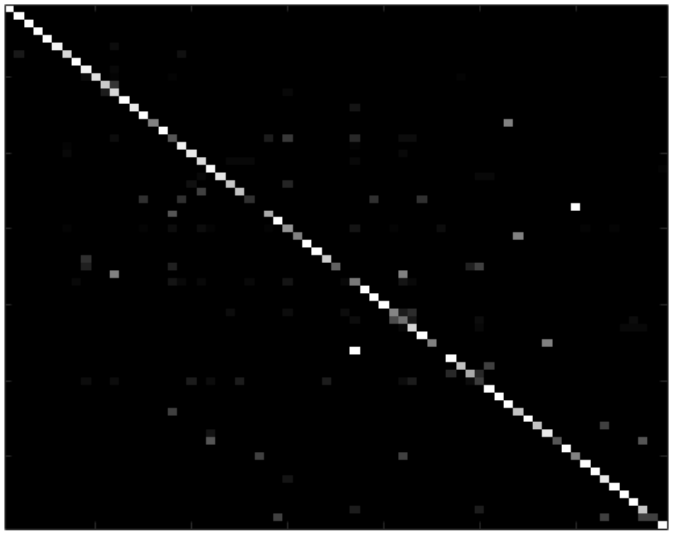


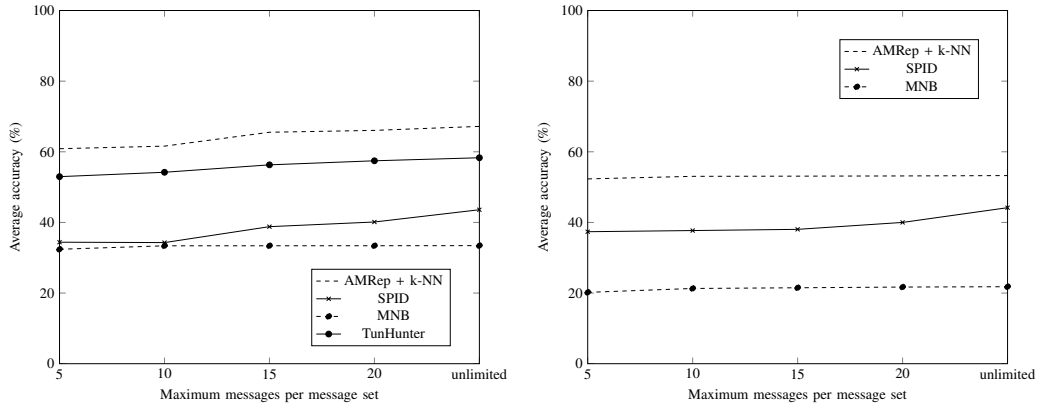
Fig. 7.3: Confusion matrix of AMRep on the Any-P dataset for all 69 applications. Each row and each column in the figure corresponds to one application and the color of each cell (i, j) represents the ratio of message sets of application i classified as application j. The lighter the color of a cell, the higher the ratio which the cell represents.

- `helpd` and `com.apple.WebKit.Networking` — `helpd` is a process running on Apple devices which connects to on-line support of Apple and downloads the help pages. WebKit is an engine for rendering web pages used, for example, by Apple Safari browser. It is likely that the `helpd` process could use this engine, too.
- `googledrivesync` and `netsession_win` — while the `googledrivesync` process is responsible for synchronizing Google Drive content between a user's computer and the on-line storage, the `netsession_win` process is part of Akamai NetSession Client which is, as stated by Akamai, "a tool to improve the speed, reliability, and efficiency for downloads and streams". While these two applications are not closely related, their basic behavioral patterns might be similar — both of them are likely transferring batches of larger data which explains why the classifier mismatched them.

Accuracy with limited number of observations

The quality of representation of probability distributions typically depends on the number of observations. To study how the AMRep representation and the prior art classification methods are sensitive to limited number of observations, the main classification experiment described in Section 7.5.2 was repeated while limiting the number of observed messages for each repeated communication to 5, 10, 15 and 20. The rest of the experimental settings remained the same as in the previous subsection and the classifiers used all features available for a given dataset. Classification accuracies of different methods are summarized in Figures 7.4a and 7.4b for Any-P and Any-F datasets. For easier comparison of the results achieved by the version of the classifier without upper bound on the number of messages per message set, these results are shown in Figures 7.4a 7.4b as well under the tick "unlimited" on the x-axis. As expected, with decreasing the limit on maximum messages used the accuracy of methods decreases. The only exception seems to be the MNB classifier which uses only the messages' sizes and is therefore less prone to errors caused but insufficient number of observations (messages). However, the AMRep representation with the k-NN classifier still

outperforms the other methods, despite that it uses joint distributions of all the features. This is because MMD metric does not need to explicitly estimate the shape of probability distributions, it can effectively operate even with very limited number of observations from each distribution. This efficiency in terms of number of observations is in fact one of the main features of MMD.



(a) Any-P dataset

(b) Any-F dataset

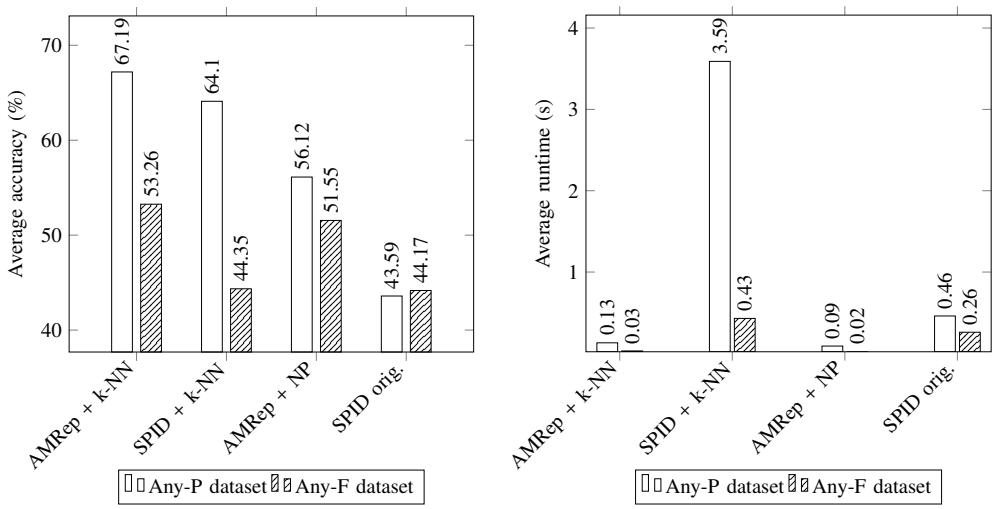
Fig. 7.4: Comparison of average accuracy depending on the maximal number of messages used from each message set.

Dependency on a classifier

The k-NN classifier is for AMRep a natural choice, since it leverages the well-defined pairwise distances between classified objects. Because the classifier uses local neighborhoods of the queried objects, it can well capture multi-modal behavior of the applications (k-NN classifier is asymptotically consistent [37]). Flows belonging to the same application (e.g., Dropbox) can belong to different modes of the application's behavior (e.g., regular polling for change notifications, files upload, filed download etc. — as presented in Chapter 5). While flows from different modes are dissimilar to each other, behavior within one mode of operation is homogenous which allows the k-NN classifier to correctly classify most of the unknown flows (this is illustrated in the visualization of the clustering of flows in the next section). However, the AMRep representation is not tightly connected with one specific classifier, as it is a general algorithm for building representations of message sets. Similarly, the SPID [72] method can be viewed as an implementation of the nearest prototype (NP) classifier paradigm [13] with Kullback-Leibler divergence as distance metric. The SPID method can be therefore extended to use the k-NN classifier with Kullback-Leibler divergence instead of the originally proposed NP classifier, while AMRep can be on the other hand used with the NP classifier by creating the applications' prototypes by merging all training messages belonging to the same application into one message set and then using the AMRep algorithm to create its representation.

This motivates the experimental comparison of these four different configurations, namely AMRep with the k-NN classifier, SPID with the k-NN classifier, AMRep with the NP classifier and the original version of the SPID method (with the NP classifier). The method are compared in the same evaluation setup as in Section 7.5.2 and the average classification accuracy and running times are measured. According to classification accuracies presented in Figure 7.5, the k-NN classifier always significantly outperforms the NP classifier, which is likely due to the multimodality of individual classes discussed above. The unimodal behavior

implicitly assumed by the NP classifier seems to be oversimplified. Furthermore, AMRep outperforms the SPID representation regardless the used classifiers. Finally, running times of classifiers that use AMRep are considerably lower, which makes them more suitable solution, especially for real-time deployment.



(a) Average accuracy (higher is better) (b) Average runtime (lower is better)

Fig. 7.5: Average accuracy and running times for AMRep and SPID used either with the k-NN classifier or with the nearest prototype (NP) classifier.

7.6 Experimental evaluation — Clustering

Clustering algorithms are essential in unsupervised analysis of unlabelled data, since they reveal groups of similar objects. One example of such application was shown in Chapter 5. In this section, we demonstrate how the AMRep representation can be used in analysis of unlabelled traffic.

7.6.1 TCP flows clustering

In this scenario, the goal is to cluster individual flows (Any-P dataset), which can be used to identify similar applications active in a network. Results of such clustering can be used, for example, to uncover new and possibly unwanted applications in the network or as a basis of anomaly detectors that discover outlying flows that do not belong to any widely used application.

Since graphical representation of clustering results enables an analyst to quickly identify groups of similar objects, we again chose to use similarity graph to visualize the results. Since AMRep is designed such that Euclidean distance between two representations approximates the MMD distance between the underlying distributions, calculating similarity with Gaussian kernel as

$$\mathrm{sim}(s_1, s_2) = \exp(-\gamma' \|s_1 - s_2\|^2), \tag{7.16}$$

corresponds to a kernel over the space of probability distributions as introduced in [25]. $s_i = \mathbf{L}^{-1}\varphi(\mu_i)$, $i \in \{1, 2\}$, in Equation 7.16 are AMRep representations of the two compared message sets as defined in Section 7.4 and γ' is the width parameter of the Gaussian kernel

on the space of representations (which is different from γ used to calculate s_i). The width parameter γ' can be used to change sensitivity of the similarity function, the experiments presented here used inverse of the median of squared distances between all pairs of representations, which is a "rule of thumb" recommended in [121]. The similarity graph is then visualized in two-dimensional plane by means of Force Atlas 2 algorithm [76], which causes highly similar vertices to be attracted to each other, while those with low similarity are repulsed. Again, to further improve visualization qualities of the graph, edges with weights close to zero can be omitted to make groups of similar vertices more distinct.

We note that the visualization algorithm and clustering were chosen to enable easy assessment by a human analyst. However, any other clustering algorithm requiring only pairwise distances, similarities, or even existence of the clustered objects in a metric space could be used. Examples include k-means, spectral clustering ([68, 69]) or the Louvain method [16], which can be even applied directly on the similarity graph and is able to optimize also the number of clusters.

The demonstration of clustering of application flows (Any-P dataset) uses a subset of 5000 TCP flows of 69 different applications used in the previous section. The subset was selected randomly, but with the restriction that there were at least 10 flows from each application. Restricting the experiment to only 5000 samples was to enable comprehensive visualization of results, as more samples would make the graph cluttered.

The similarity graph drawn as described above is shown in Figure 7.6. Nodes are colored according to different applications (we recall that ground truth is known in this case) to which the respective flows belong. Flows (nodes) belonging to same applications mostly form well separated clusters which asserts that AMRep representation is able to distinguish the applications, using only information about packets' sizes and their inter-arrival times.

This visualization also helps to understand why the k-NN classifier outperforms the NP classifier, as discussed in Section 7.5.3. For example, the application "Cisco Jabber" is spread over multiple clusters which indicates that it has multiple types of behavior. However, each cluster is homogenous which allows the k-NN classifier to correctly classify flows from different clusters (it uses the k nearest neighbors that will be likely from the same cluster). As in previous section, some clusters contain multiple applications mixed together. A closer inspection of flows of these clusters shows that they have indeed something in common. Few larger clusters with mixed application contain:

- `com.apple.WebKit.Networking`, `WebProcess` and `Google Chrome`, which can be explained by WebKit (as already mentioned in the previous section) being a widely used rendering engine for web pages and being used as a component of many other applications including the Chrome browser.
- `Dropbox.exe`, `DropboxOriginal` and `Dropbox109` are different names of the Dropbox application running on different operating systems that appeared in the network from which the data were collected.
- `Mail`, `CalendarAgent` and `AddressBookSourceSync` are all processes running on Apple's devices responsible (as their names suggest) for a mail client operation and the Calendar application. These two applications frequently work in tandem, e.g., when a user connects to Microsoft Exchange server which manages e-mails, meeting invitations etc.

To further review the results of AMRep in a clustering task, the Louvian clustering algorithm [16] was applied to the same 5000 TCP flows, using the same similarity function and graph as used in visualization in Figure 7.6. The algorithm returned 73 clusters, which is surprisingly close to the true number of different applications in the data, which was 69. Furthermore, the same similarity graph was constructed for similarities determined by FastMMD and also for soft histogram representations (in which case the cosine similarity was used as it performed better in experiments in Chapter 5) and Louvian clustering was applied on these graphs, too. Results of these three clusterings are compared by means of the ARI index [114] in Table 7.4. As we can see, the results follow the trend observed in the

previous section — representation based on kernel embedding, especially AMRep, provide more accurate results when compared to soft histogram ones.

Representation	ARI
AMRep	0.71
FastMMD	0.68
SoftHist	0.66

Table 7.4: Values of Adjusted Rand Index (ARI) for clustering of 5000 TCP flows (Any-P) dataset based different representations.

Although the ARI value 0.71 might seem to be low (given that the ideal solution should achieve 1.00), an inspection of applications in 8 clusters with the highest silhouette score [117] (shown in Table 7.5) reveals that the real number will be higher, because of multiple applications with different labels are in fact very similar — often just different variants of the same application. For example, Dropbox clients running on different platforms, as can be seen in Figure 7.6, or the cluster number 5 in Table 7.5 containing only flows belonging to applications `Meeting Center` and `atmgr.exe` that are both related to Cisco Webex service, or the cluster number 8 which contains mixture of Cisco Jabber and Cisco Webex/Meeting Center applications that can all work integrated together to allow communication and organizing calls or teleconferences.

7.7 Chapter summary

This chapter presented an alternative approach to representing network communication based on kernel embedding of probability distributions. The proposed representation is derived from a well defined distance on spaces of probability distributions used in kernel two-sample hypothesis test (known as Maximum Mean Discrepancy). The main advantage of this representation is that it does not need to estimate the shape of the probability density functions to compare the distributions which enables modeling multivariate distributions of higher dimension even from limited number of observations.

The representation was evaluated and compared to the prior art in the supervised setting on the problem of identification of application from the observed traffic at two different levels: (i) identification of application from a set of measurements on TCP packets and (ii) identification of application from a set of flows that a client exchanged with a server. The representation was also tested in an unsupervised scenario of identifying groups of TCP flows initiated by the same applications. The experimental results showed promising potential of the representation also in this area.

Additionally, the representation was also compared to the soft histogram representation described in Chapter 5. While the AMRep representation was able to achieve better accuracy in classification of the network traffic, the soft histograms are on the other hand able to produce sparser representations. Therefore, both approaches can find their utilization in practice. When the amount of data to be processed is smaller and the accuracy is very important, the AMRep should be preferred. When the scalability and fast processing of large volumes of data (e.g., as a pre-filtering step in more complex security solutions) plays the most important role, the soft histogram representations can be successfully deployed.

Finally, the approximate computation of distances between probability distributions proposed in this chapter is not limited to the domain of network traffic modeling. It can be considered for any problem in which the analyzed objects are samples from probability distributions. This includes multi-instance learning problems (MIL) or Support Measure Ma-

Cluster	Application	Percentage
1	Box Sync	100.00%
2	Dropbox.exe	50.00%
	DropboxOriginal	50.00%
	thunderbird.exe	89.33%
	vmnat.exe	3.37%
	Meeting Center	3.00%
	vmware-vmrc.exe	2.00%
3	ssh	100.00%
4	APSDaemon.exe	99.49%
5	Meeting Center	55.56%
	atmgr.exe	44.45%
6	com.apple.WebKit.Networking	89.47%
	Google Chrome	5.26%
	firefox.exe	5.26%
7	Dropbox.exe	50.00%
	Dropbox109	12.50%
	DropboxOriginal	12.50%
	Cisco Jabber	12.50%
	Google Chrome	12.50%
8	Cisco Jabber	45.45%
	Meeting Center	18.18%
	Cisco Webex Start	15.15%
	CiscoJabber.exe	9.09%
	com.apple.WebKit.Networking	6.06%

Table 7.5: Example of 8 clusters with the highest silhouette score produced by the Louvain algorithm applied on a similarity graph created using AMRep representations of data used in Section 7.6.1. For each cluster, applications with at least 1% representation in the cluster are presented.

chines [96]. To better illustrate this, Appendix of this thesis includes an experiment which shows that AMRep can be used to speed-up a Support Measure Machine classifier.

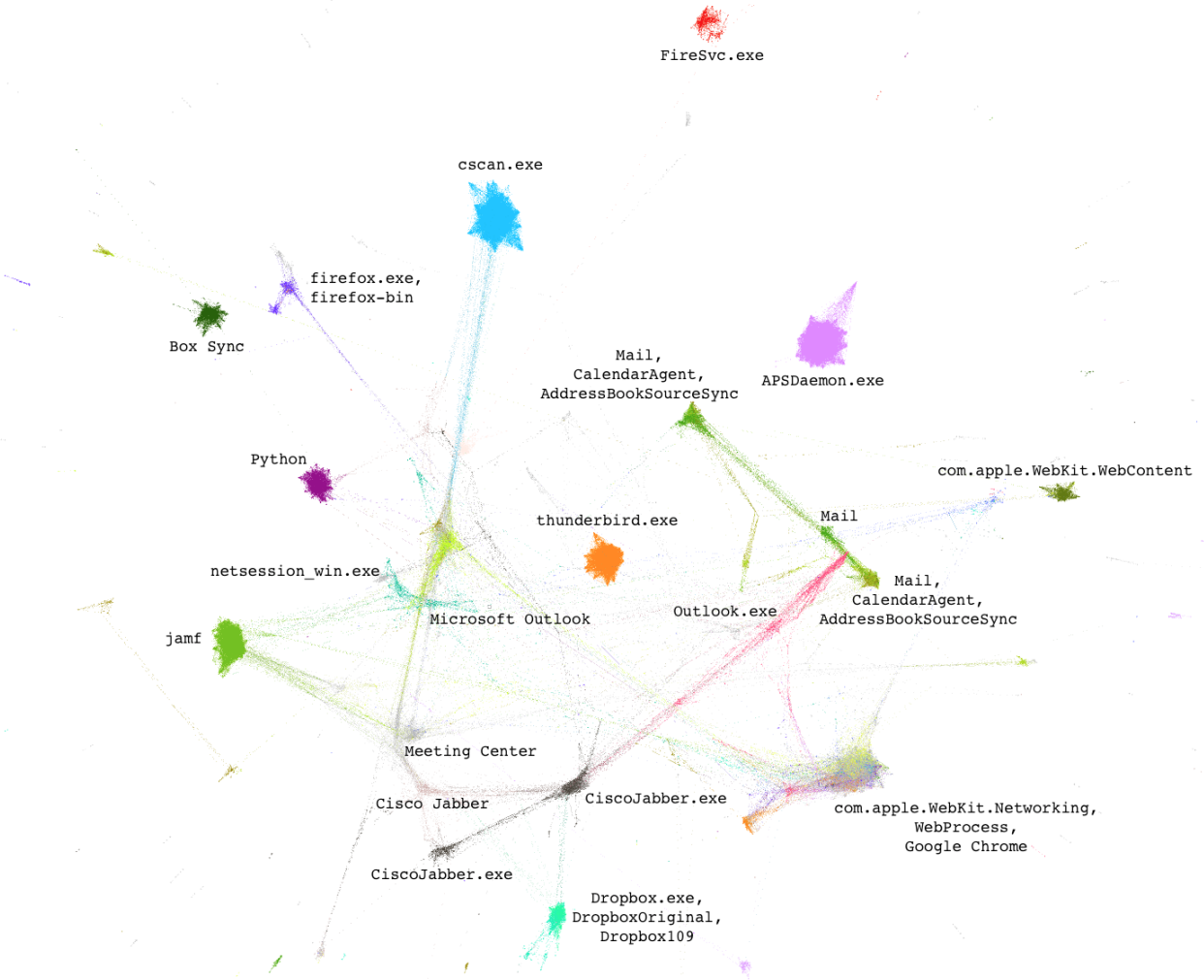


Fig. 7.6: Visualization of similarities of TCP flows (each treated as a message set of packets) belonging to different applications. Each node represents one TCP flow. The vertices are colored according to the ground-truth application labels contained in the *TCP Flows* dataset. Clusters of nodes containing the most prevalent applications are labelled with those applications' names.

8

Dictionary representations of persistent behavior

In previous chapters, we studied representations based on the formal model proposed in Chapter 3 which assumed that the messages interchanged in the communication can be represented as points in a d-dimensional real space. In other words, the representations expected that all the features observed on the messages are real-valued. However, in certain cases, behavior of a network entity can be characterized also by features that do not come from a continuous domain with ordering. To illustrate this, let us consider following examples:

- It can be expected, that a server providing a specific service can be very likely identified based on the distribution of transport layer port numbers that it uses. At the transport layer, the service can be identified by its assigned port number. Although the server can also communicate on other ports because of other applications running on it (e.g., operating systems updates, remote logins for administration purposes and server maintenance and similar purposes), if the main role of the server is to provide that specific service, its assigned port can be expected to be dominant over other ports observed as active on the server. While, of course, a service can be configured to use some non-standard port for its activity which would make this approach ineffective, in many cases this relatively simple heuristic of marching by port is still successful in discovery of network assets.
- Similarly, user's identity can be characterized by a set of web pages that he or she frequently visits. Naturally, a user will likely visit many different sites during a period of a day or two. However, many of these visits will be emphemeral as they are more or less random visits not tightly connected with user's identity. On the other hand, users tend to have their favorite sites that they check repeatedly for new content and updates. A user will likely check his/her favorite social networks, read news published on news sites that he or she likes or even use other services that utilize HTTP communication and communicate to specific servers and domains (e.g., cloud storage services like Box, Dropbox or Amazon S3, cloud computing services like Amazon Web Services and similar platforms). The combination of these repeatedly contacted sites can be very specific for the given user and, as such, can be used to model baseline of user's normal behavior to detect anomalies. These approaches form foundations of User and Entity Behavior Analytics [55, 77, 122, 82, 60] that are being developed as a protection against so called insider threats.
- A device, like a cell phone, tablet, laptop or desktop PC can be identified and profiled, for example, by distribution of UserAgent strings observed in its HTTP communication [65]. The User-Agent field passed in headers of the HTTP protocol is designed to identify the client's application which is requesting the given resource from a web server. Therefore, it may contain a lot of information characterizing the application and the device on which it is running. For example, in case of web browsers, it typically contains the browser's name, version, version and type of the browser's core and other information that even allow to infer the device's operating system from it. By observing the User-Agent strings that are repeatedly reported from the given device its profile can be built and used, for example,

to track the device over different IP addresses (if no other identifiers are available) or to detect suspicious changes in its behavior (which, again, falls into the UEBA models mentioned above).

All the above listed examples show use-cases in which discrete features are used to model long-term (or persistent) behavior of a user or device. This discrete (or categorical) nature of the features does not allow to directly represent the messages in a communication as points in a d-dimensional real space and directly apply the representations described in previous chapters.

While approaches based on string kernels [88] or semantic word embedding [93] could be in theory used, in practice it is problematic due to unclear semantic relations and similarities within features. Moreover, whenever a new feature of this type appears, the embedding mechanism needs to be trained for it. Therefore, more lightweight ways how to represent the network entities' behavior using such features need to be found. An easier to maintain way is to use an approach inspired by the Bag-of-Words (BoW) representations applied often in natural image processing, text analysis or even computer vision [135]. In this case, the definition of a message from Chapter 3 can be further relaxed such that a message m is represented as an instance from some dictionary $\mathcal{D}$ of possible feature's instances. The BoW representation is a frequency vector in which each element represents one possible instance from the dictionary and the value of the respective element represents the frequency of observing this specific instance for the given object that is being represented.

To illustrate this by an example, let us assume that the messages are contacted servers' hostnames (hence, the dictionary $\mathcal{D}$ contains all hostnames that can be possibly contacted). The observed instances can be, for example, `www.google.com`, `mail.google.com`, `www.yahoo.com` etc. If a user (whose behavior is the represented message set) issued, for example, two requests to `www.google.com`, two requests to `mail.google.com` and one request to `www.yahoo.com` within the time frame in which the observations are collected, then the resulting representation looks like:

$$(\texttt{www.google.com} : 2, \texttt{mail.google.com} : 2, \texttt{www.yahoo.com} : 1)$$

Many different variants of the BoW model have been proposed in the literature, focusing on different aspects of the representation, including automatic learning of the dictionary or enforcing sparsity of representations ([130, 152, 149, 92]). The produced representations are potentially high-dimensional but sparse vectors that can be thus effectively processed using similar algorithms as shown in Chapters 5 and 6, as they can be treated as histogram representations, too. However, a problem of this straightforward approach shown in the above example is that if the pattern which is the most important for the given message set is not prevalent enough, it might be easily suppressed by other communication which is not that specific for the represented entity. Consider, for example, a device whose operating system is periodically checking for updates on its vendors' web server. These regular checks for updates are a good indicator of what type of device it is and what kind of operating system it is running. On the other hand, if the device is not left idle and is used for common work, much more communication activity can be observed from it which will overwhelm the relatively lightweight activity related to that update checks. This hardens the correct inference the device type. Therefore, we further propose a representation which is inspired by the idea of the BoW model but is designed to capture *persistent* behavior of the modeled entity and to suppress the influence of high but short-term bursts of communication that could hide the important patterns specific for the given entity.

8.1 Modeling of persistent behavior

As was stated above, for cases when it is desired to capture long-term (which we call persistent) patterns in behavior of modeled entities, the BoW model needs to be properly modified.

The modification which we propose is distantly inspired by the method of measuring persistence published in [59]. The core idea is not to capture raw frequencies of the observed messages, but to capture in how many different time windows the message was observed. This approach can be viewed as a kind of de-noising. While the raw counts of message's observations can significantly vary even for the same entity, the number of time windows in which a message was observed will tend to be more stable if a given message indeed belongs to a persistent behavioral pattern. Formally, the method can be described as follows:

Let us consider a dictionary of possible messages' instances $\mathcal{D}$ and a message set $\mathcal{R} = \{m_1, ..., m_n | m_i \in \mathcal{D}\}$ which captures behavior of the represented entity. Additionally, we assume that for each message m_i, we have a timestamp indicating the time when the message was observed, denoted as t_{m_i}. This is not a constraint in practice, as most of the logging systems (such as routers or web proxies) provide a timestamp for each log entry anyway. The time period for which the representation is built is divided into T consecutive time windows $\mathcal{W} = \{w_1, ..., w_T\}$ of the same length L. The values T and L are parameters of the representation and can be set arbitrarily with respect to the given situation. For example, if L is set to 1 hour and T to 24, then the representation will cover 1 day of behavior composed of 24 time windows, each spanning 1 hour.

The frequency vector h representing the given message set is then constructed in the similar way as in the standard BoW model but the value for an entry $m \in \mathcal{D}$ is given by the formula:

$$h_m = \sum_{i=1}^{T} [\ \exists m \in \mathcal{R} : t_m \text{ is within } w_i\ ]. \tag{8.1}$$

Instead of counting the raw occurrences of $m \in \mathcal{D}$ in the message set $\mathcal{R}$, the proposed representation counts the number of windows in which the message m was observed. This suppresses the effect of messages that would appear as bursts within a small number of time windows.

The above described process of building a representation of persistent behavior using the modified BoW model is illustrated in Figure 8.1. In this example, the dictionary $\mathcal{D}$ consists of servers' hostnames. In the upper part of the figure, we can see the message set $\mathcal{R}$ for which the representation is built, containing hostnames that were contacted by the modeled entity (e.g., a client's personal laptop) together with timestamps indicating when the connections to those servers were issued. Then, in the middle part of the figure, we can see the time windows $\mathcal{W}$ and values of the Iverson bracket term used in Equation (8.1) (indicated by 0/1 values). The example uses 6 windows ($T = 6$) of length 5 minutes ($L = 5$ minutes). Finally, in the bottom part of the figure, we can see the BoW frequency vector created by summing the results of the predicate for each message $\mathcal{D}$.

8.2 Experimental evaluation

In this section, we evaluate the method for representation of persistent behavior using discrete features described in this chapter. We show that by using the proposed representation, an operating system type can be classified using only information about hostnames of servers that were contacted by the classified device. The dataset *Hostnames* was used for this evaluation which is described in Chapter 4 (Section 4.5).

8.2.1 Operating system family classification

Motivation

An administrator of a corporate network is typically able to identify a device type or operating system of only a fraction of the network connected devices that have manually assigned static IP addresses or IP range and serve some specific purpose. However, devices in the

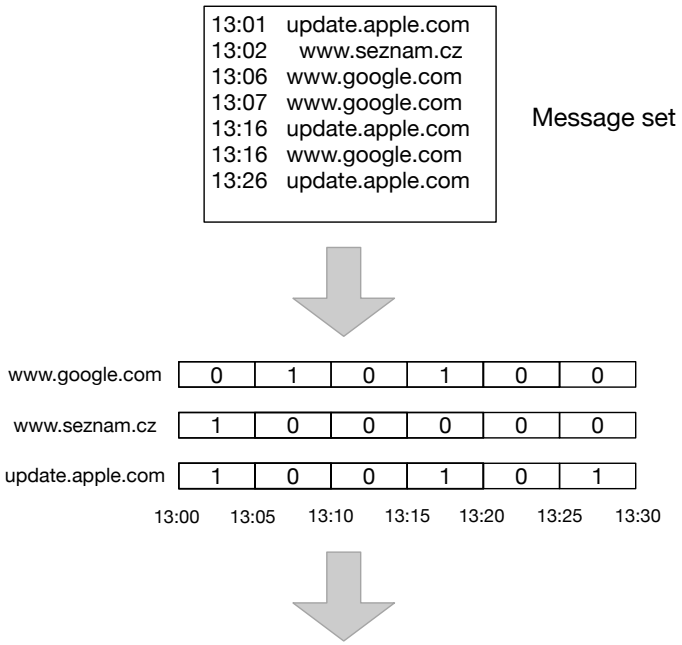


Fig. 8.1: Example of building a BoW representation which captures persistent behavior of the modeled entity.

dynamically assigned ranges (e.g. guest networks, campus networks, bring-your-own-device (BYOD) networks, etc.) cannot be tracked this way. However, logs of network activity of a device can contain information which helps to identify its kind. Among other activities, an operating system occasionally contacts specific servers and services, for example, to check the Internet connection, to check and download software updates and applications from repositories or to report technical issues. Therefore, logs of these connections can be leveraged to identify what type of operating system is running on an unknown device. This helps an administrator to get insight into the network and might help to identify devices that are not allowed in the network or to identify potential risks.

Network traffic logs were utilized in prior art works to identify the devices. User-Agent strings were leveraged in several earlier works to identify operating systems. The work [3] used the User-Agent strings together with additional features extracted from encrypted TLS communication to classify operating systems. Similarly, the work [75] uses a dictionary of fingerprints of initial packets in TLS connections and User-Agent strings to identify client devices (their types including operating systems). The User-Agent strings are used as labels to identify an unknown device when only the TLS fingerprints are available from it.

Unlike User-Agent strings or parameters of TLS connections, information about the destination of a connection — the target server — can be obtained from almost all logs of communication. Hostname of the target server can be either logged directly (for example, in case of web proxy logs) or it can be inferred from the IP address of the server using passive DNS records. Despite that, to best of our knowledge, there is no prior art work that would deal with devices identification based purely on the logs of contacted servers.

Therefore, this experiment had two main goals. First, to verify that even the relatively limited information in the form of contacted hostnames can be used to identify the types of operating systems. Second, to show that the performance of operating systems classification improves with the increasing number of time windows that are used to capture the behavior of devices, because the modified BoW model focuses on persistent behavior which we assume

is more distinctive after longer period of observation. Additionally, the experiment shows how long a device's activity needs to be observed in order to achieve good classification performance.

Dictionary

The *Hostnames* dataset uses hostnames of contacted servers as individual messages. Therefore, the dictionary $\mathcal{D}$ could be a set of all possible hostnames in the Internet, which would be huge. Because of that, we employed the following heuristic selection of hostnames for the dictionary in order to make it more compact and suitable for processing.

A large corpus of HTTP requests coming from 500 corporate networks and spanning one week of traffic was obtained (we emphasize that the three companies from which the dataset *Hostnames* was created were not included in those 500). Then, we selected requests that were identified as belonging to devices running one of the three operating systems contained in the *Hostnames* dataset (i.e., Apple, Android or Windows) by using the information contained in the User-Agent strings when available as in [75].

Next, we uniformly sampled from the corpus equal number of requests for each operating system family, namely 3 000 000 requests for each. These sampled data were then used to estimate a conditional probability $P(os|h)$ for each hostname h and operating system family os in the sampled data, i.e. conditional probability distributions over the three operating system families conditioned on each hostname. The motivation for this is that if the conditional distribution has a very low entropy, the given hostname h is a good candidate for the dictionary as it is possibly indicative for one of the operating system families.

Finally, the hostnames were sorted in ascending order by entropies of their respective conditional distributions to reveal the most indicative ones and 120 hostnames were selected after manual analysis for the dictionary. Therefore, the dictionary $\mathcal{D}$ used for this experiment contained 120 selected hostnames.

Classification

The experiment simulates the case when there are labeled data available to build a classifier which is then able to classify representations of devices in an unknown network (or its part) for which the labels can not be obtained. The evaluation was performed in the "leave-one-company-out" manner. In each fold, data from networks of two companies from the dataset were used as labelled data for building a classifier and data from the third company were used for evaluating performance of the classifier. The final results were then averaged over all three folds. This setup well simulated the case when a trained classifier is deployed in a new unknown network.

The parameter L (time period covered by one time window), which is the size of one time window, was set to 5 minutes. The parameter T (i.e., the number of consecutive time windows used to build the representation) varied from 96 (to cover 8 hours of behavior) to 336 (to cover 28 hours of behavior).

To provide the first insight into the problem, we present a t-SNE [137] plot created using a sample of data from these three companies in Figure 8.2. Each dot represents one device and distances between dots follow L_2 distances between the respective BoW vectors that represent the devices. Dots are colored with respect to the operating system family determined by the ground truth label contained in the dataset. The plot shows that significant portion of devices form smaller and clear clusters of those that are running the same type of operating system. On the other hand, the total composition of the dots — especially in the middle region — promotes usage of a classifier which would leverage the local similarities. This, therefore, leads us to apply the k-NN classifier again which is exactly able to leverage these properties.

The value of parameter k in the k-NN classifier was in each fold of the "leave-one-company-out" evaluation determined by cross-validation performed on data from the two companies that were used as the training set in the given fold. The searched values of k ranged from 1

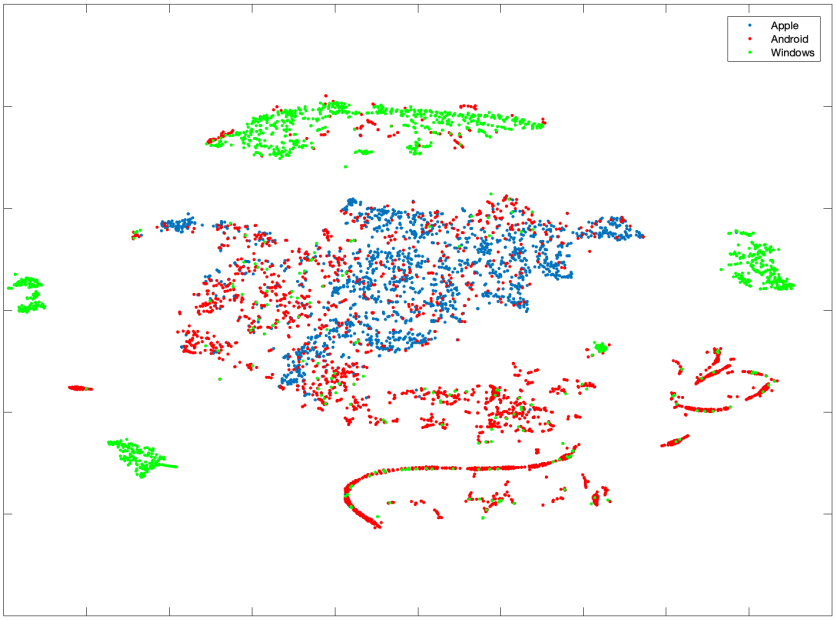


Fig. 8.2: Visualization of similarities of different operating system families (Windows class was uniformly sampled).

to 20. The k-NN classifier used the L_2 distance function and the majority voting among the nearest neighbors was used to obtain the classifier's output.

Results of the classification are shown in Figure 8.3. We present classification precision and recall for individual operating system families depending on the parameter T (the number of time windows used). First, as can be seen from the graphs in Figure 8.3, the results prove that basic identification of operating systems using only the information about contacted servers is possible. Furthermore, we can see that as the parameter T increases and the representation covers longer time period, both the precision and recall show rising trends, too. This confirms the assumption that as we observe the behavior of the devices for a longer time, the behavioral patterns specific for individual operating system families emerge more clearly and the representation of the behavior is able to capture them. Differences in classification performance of individual operating systems exist — the most is successful is the identification of Windows-based systems (which is almost perfect), mainly because their behavior is very specific with respect to the persistently contacted domains hosted by Microsoft (the clear separation of the devices can be seen from the plot in Figure 8.2).

On the other hand, the lowest precision and recall was achieved on the Android-based systems. This can be attributed to the fact that in case of Android systems, there are multiple different vendors that provide more or less modified variants of the system. Therefore, behavior of the Android-based devices is more heterogenous which decreases the accuracy of their identification.

Overall, we can conclude that after approximately one day of traffic (i.e., for $T > 288$), the operating systems can be identified with reliable results and can be considered for practical deployments.

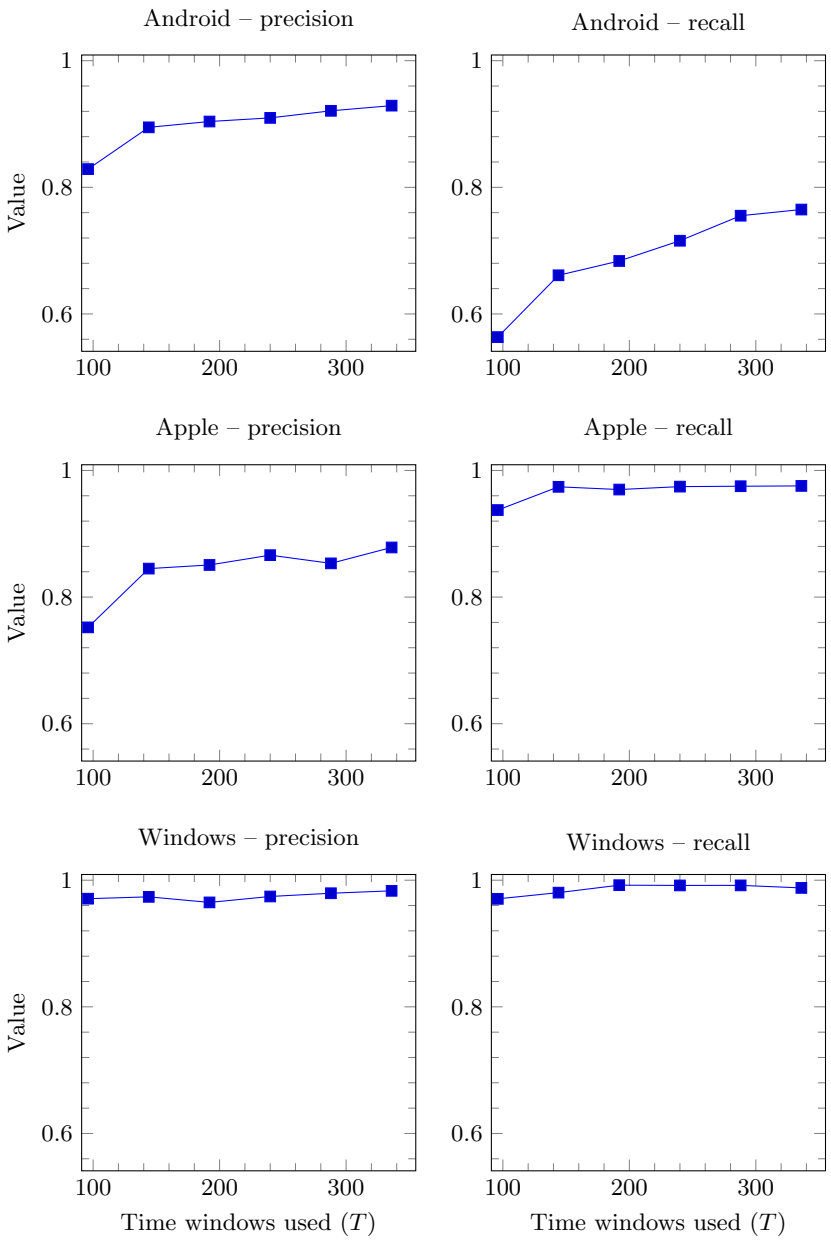


Fig. 8.3: Precision and recall of OS family classification for different operating system families depending on the number of time windows used. The trend shows that with increasing number of time windows, performance of the classification increases as well. The representation is able to capture long-term patterns in the devices' behavior that are specific for different types of operating systems.

8.3 Chapter summary

In this chapter, we proposed a representation which is able to process messages represented by discrete features. Specifically, we focused on representation of persistent patterns in the behavior of modeled entities. The representation is derived from the Bag-of-Words (BoW) model and adapted to emphasize messages that appear repeatedly in the communication, but not necessarily with regular or periodical pattern.

We demonstrated usage of this representation on the task of identification of operating system families running on network devices. The results show that by using the BoW-based representation of persistent behavior, the high-level identification of operating systems is possible using only very limited information about their activity — hostnames of contacted servers. Such information can be obtained from almost all types of traffic logs with minimal impact on users' privacy. This makes this method very sufficient for various scenarios in which insight into a network needs to be obtained.

9

Conclusions

The applications of machine learning algorithms for analysis of network traffic data, either for security or network management purposes, became common in recent years. Main reasons why we can see such trend include both the rising amounts of data that are infeasible to handle by humans and more successes of artificial intelligence in other domains. With wider adoption of the algorithms that help to process the data, different representations of the analyzed traffic were developed as well. In this thesis, we reviewed various representations used in prior works to automatically detect malware activity or to identify applications communication over the network.

While the reviewed representations were mostly proposed for specific tasks and algorithms, some common patterns can be observed. Based on these observations, we introduced the general notions of messages and message sets that represent the basic units of communication. Using this terminology, we proposed a formal model of communication which is independent of any specific representation or algorithm. This model was based on probabilistic view and provided an unifying view on the representations and a basis from which the representations can be derived.

Using this formal model, we described different approaches for representing the network traffic as samples form probability distributions. While the histogram-based approaches offer a scalable way how to capture the behavior of network entities, the approaches based on kernel embedding of distributions provide improved accuracy and possibility to jointly model higher number of features.

The representations were evaluated in different scenarios that cover varied use-cases in network security and management. The key motivation was to propose representations that can be used by computer algorithms which significantly ease the analysis of data by human analysts. Therefore, in each representation we put emphasis on the ability to easily compare the represented objects by appropriate similarity functions. As we stated in the introduction, the ability to find similar or dissimilar objects in a dataset is a key precondition for various tasks in data analysis — the anomaly detection, clustering or classification of objects. Therefore, we demonstrated in the experiments various scenarios that all leverage the pair-wise similarities or dissimilarities of the representations to achieve the desired goals. This was mostly demonstrated on the k-nearest neighbors (k-NN) paradigm that exactly relies on the properties of similarity function. In the security domain, data are often very complex and the local similarities have to be leveraged which is exactly done by the k-NN classifier.

Similarly, in tasks of unsupervised analysis data like clustering, the pair-wise similarities play critical role, too. If the representations do not provide reliable way how to assess the similarity of two objects or such similarity function does not reflect the intuition which entities should be grouped together and which not, all the algorithmic machinery for processing the representations would be useless.

Overall, the proposed representations proved to be general frameworks that can be adapted to different use-cases. We showed that they can be easily suited to model network

traffic at different layers of the TCP/IP stack and used in both supervised and unsupervised machine learning applications. This represents an important contribution to the field of artificial intelligence application in network traffic analysis.

The approach established in this thesis can be used in future works to further improve and extend the ways how to represent behavior in computer networks and learn new representations. Moreover, the results achieved in the domain of kernel embedding of distributions go beyond the scope of network security as they can be applied in other domains as well, including general learning with probability distributions or multi-instance learning (MIL) problems.

9.1 Thesis achievements

Here we summarize the main accomplishments achieved in this thesis. The list also contains references to respective chapters in which the main part of each contribution is discussed:

- First, after reviewing various representations of network traffic in prior art works, a formal model of the communication was proposed. The model is based on probabilistic approach, viewing each sample of communication as sample from a probability distribution. (Chapter 2 and Chapter 3)
- Next, histogram-based approaches for representing the traffic behavior were evaluated in different scenarios and an effective algorithm for k-NN similarity search on large corpus of histogram data was proposed and evaluated. This demonstrated that the sparse joint histograms are representations suitable for large scale analysis of network data. (Chapter 5 and Chapter 6)
- Furthermore, representations based on kernel embedding of probability distributions were explored. A new representation which allows theoretically well justified comparison of represented objects based on approximated Maximum Mean Discrepancy (MMD) was designed, because MMD was originally proposed as a two-sample test (test of equality of two distributions) which has also properties of a metric on the space of probability distributions.

  The experiments with algorithms that directly utilize this information about similarity or dissimilarity of objects proved that the proposed representation can be successfully deployed in both supervised and unsupervised analysis of data. Moreover, the results achieved in this area can be extended outside the scope of network security to other domains where the objects of interest are samples from probability distributions, for example, support measure machines (SMM).

  The proposed method represents each sample from a probability distribution in a form of real vector such that Euclidean distances between vectors approximate the original MMD. Thanks to this property, a large variety of algorithms can be immediately applied on the representations regardless of whether they were originally designed for analysis of probability distributions or not. (Chapter 7)
- Finally, a modified Bag-of-Words (BoW) model was proposed to represent persistent behavior of modeled entities using discrete features. Usability of this representation was demonstrated on a task of identifying operating system families by observing contacted servers. The results proved that by leveraging the long-term information about the behavior, operating system families can be identified using only very basic information about their behavior. (Chapter 8)

9.2 Author's publications

The sections summarizes the publications of the author of this thesis:

Articles in journals with impact factor

1. **Jan Kohout**, Tomáš Pevný. Network Traffic Fingerprinting Based on Approximated Kernel Two-Sample Test. In: *IEEE Transactions on Information Forensics and Security.* 2017, 13(3), pages 788–801. Impact factor 6.21 (80%)

2. **Jan Kohout**, Tomáš Komárek, Přemysl Čech, Jan Bodnár, Jakub Lokoč. Learning communication patterns for malware discovery in HTTPs data. In: *Expert Systems with Applications.* 2018, 101, pages 129–142. Impact factor 4.92 (20%)

3. Ján Jusko, Martin Rehák, Jan Stiborek, **Jan Kohout**, Tomáš Pevný. Using Behavioral Similarity for Botnet Command-and-Control Discovery. In: *IEEE Intelligent Systems.* 2016, 31(5), pages 16–22. Impact factor 2.59 (15%)

Patents

1. **Jan Kohout**, Ján Jusko, Martin Rehák, Tomáš Pevný. Detection of malicious network connections. US Patent 9,531,742, 2016. (Issued)

2. **Jan Kohout**, Tomáš Pevný. Framework for joint learning of network traffic representations and traffic classifiers. US Patent 10,079,768, 2018. (Issued)

3. Jan Mrkos, Martin Grill, **Jan Kohout**. Tracking users over network hosts based on user behavior. US Patent 10,129,271, 2018. (Issued)

4. Martin Kopp, Martin Grill, **Jan Kohout**. Identifying self-signed certificates using http access logs for malware detection. US Patent 10,375,097, 2019. (Issued)

5. **Jan Kohout**, Tomáš Pevný. Statistical fingerprinting of network traffic. US Patent Application 15/409,746, 2019. (Allowed by USPTO, issue pending)

In ISI proceedings

1. **Jan Kohout**, Tomáš Pevný. Unsupervised detection of malware in persistent web traffic. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1757–1761, IEEE, 2015. (50%)

2. **Jan Kohout**, Tomáš Pevný. Automatic discovery of web servers hosting similar applications. In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 1310–1315, IEEE, 2015. (50%)

3. Přemysl Čech, **Jan Kohout**, Jakub Lokoč, Tomáš Komárek, Jakub Maroušek, Tomáš Pevný. Feature extraction and malware detection on large HTTPS data using MapReduce. In: *International Conference on Similarity Search and Applications*, pages 311–324, Springer, 2016. (16.67%)

4. Martin Kopp, Martin Grill, **Jan Kohout**. Community-based anomaly detection. In: *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, IEEE, 2018. (20%)

In other proceedings

1. Jakub Lokoč, **Jan Kohout**, Přemysl Čech, Tomáš Skopal, Tomáš Pevný. k-NN Classification of Malware in HTTPS Traffic Using the Metric Space Approach. In: *Pacific-Asia Workshop on Intelligence and Security Informatics*, pages 131–145, Springer, 2016. (20%)

9.3 Other author's publications

Publications of the author not directly related to the topic of this thesis:

In ISI proceedings

1. **Jan Kohout**, Roman Neruda. Two-phase genetic algorithm for social network graphs clustering. In: *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, pages 197–202, IEEE, 2013. (50%)

In other proceedings

1. **Jan Kohout**, Roman Neruda. Exploration and exploitation operators for genetic graph clustering algorithm. In: *International Symposium on Methodologies for Intelligent Systems*, pages 87–92, Springer, 2012. (50%)

References

1. C. C. Aggarwal. *Outlier Analysis*. Springer New York, 2013.
2. M. Ahmed, A. Naser Mahmood, and J. Hu. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 2016.
3. 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)*, pages 1–9. IEEE, 2017.
4. N. Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, 1950.
5. F. Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.
6. F. R. Bach and M. I. Jordan. Predictive low-rank decomposition for kernel methods. In *Proceedings of the 22Nd International Conference on Machine Learning*, 2005.
7. M. Bailey, E. Cooke, F. Jahanian, Y. Xu, and M. Karir. A survey of botnet technology and defenses. In *2009 Cybersecurity Applications & Technology Conference for Homeland Security*, 2009.
8. F. Baker. Requirements for IP Version 4 Routers. RFC 1812, 1995.
9. M. Bastian, S. Heymann, and M. Jacomy. Gephi: An open source software for exploring and manipulating networks, 2009.
10. L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state Markov chains. *The annals of mathematical statistics*, 1966.
11. R. Bayer and E. McCreight. Organization and maintenance of large ordered indexes. In *Software pioneers*, pages 245–262. Springer, 2002.
12. L. Bernaille, R. Teixeira, and K. Salamatian. Early application identification. In *Proceedings of the 2006 ACM CoNEXT Conference*, 2006.
13. J. C. Bezdek and L. I. Kuncheva. Nearest prototype classifier designs: An experimental study. *International journal of Intelligent systems*, 2001.
14. L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel. Disclosure: Detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012.
15. A. Bjerhammer. Application of calculus of matrices to method of least squares with special reference to geodetic calculations. *Transactions of the Royal Institute of Technology, Stockholm, Sweden*, 1951.
16. V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008.
17. J. Brabec and L. Machlica. Decision-forest voting scheme for classification of rare classes in network intrusion detection. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2018.
18. R. T. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122, 1989.
19. S. Brin. Near neighbor search in large metric spaces. In *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland.*, pages 574–584, 1995.
20. A. H. Cannon and L. J. Cowen. Approximation algorithms for the class cover problem. *Annals of Mathematics and Artificial Intelligence*, 2004.

21. S.-H. Cha. Comprehensive survey on distance/similarity measures between probability density functions. *International Journal of Mathematical Models and Methods in Applied Sciences*, 2007.
22. Ch.-Ch. Chang and Ch.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.
23. Edgar Chávez and Gonzalo Navarro. A compact space decomposition for effective metric indexing. *Pattern Recogn. Lett.*, 26(9):1363–1376, July 2005.
24. Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
25. A. Christmann and I. Steinwart. Universal kernels on non-standard input spaces. In *Proceedings of the 23th International Conference on Neural Information Processing Systems*, 2010.
26. O. Chum. Low dimensional explicit feature maps. In *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.
27. O. Chum, J. Philbin, and A. Zisserman. Near duplicate image detection: min-hash and tf-idf weighting. In *BMVC*, 2008.
28. Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB'97*, pages 426–435, 1997.
29. Cisco. Cisco AnyConnect Secure Mobility Client. `https://www.cisco.com/c/en/us/support/security/anyconnect-secure-mobility-client/tsd-products-support-series-home.html`, 2019.
30. Cisco. Cisco Tetration. `https://www.cisco.com/c/en/us/products/data-center-analytics/tetration-analytics/index.html`, 2019.
31. J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, 1965.
32. M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Detecting http tunnels with statistical mechanisms. *Proceedings of the 42nd IEEE International Conference on Communications (ICC)*, 2007.
33. M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Traffic classification through simple statistical fingerprinting. *SIGCOMM Comput. Commun. Rev.*, 2007.
34. A. Dainotti, F. Gargiulo, L. I. Kuncheva, A. Pescapè, and C. Sansone. Identification of traffic flows hiding behind tcp port 80. In *2010 IEEE International Conference on Communications*. IEEE, 2010.
35. J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
36. S. E. Deering and B. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 8200, 2017.
37. L. Devroye, L. Györfi, and G. Lugosi. *Consistency of the k-Nearest Neighbor Rule*, chapter 11, pages 169–185. Springer New York, 1996.
38. I. Drago, M. Mellia, M. M. Munafo, A. Sperotto, R. Sadre, and A. Pras. Inside Dropbox: Understanding personal cloud storage services. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, 2012.
39. P. Drineas and M. W. Mahoney. On the Nyström Method for Approximating a Gram Matrix for Improved Kernel-Based Learning. *Journal of Machine Learning Research*, 6:2153–2175, 2005.
40. R.P.W. Duin, P. Juszczak, P. Paclik, E. Pekalska, D. de Ridder, D. M. J. Tax, and S. Verzakov. PR-Tools, a matlab toolbox for pattern recognition, 2015. `http://prtools.org`.
41. M. Dusi, M. Crotti, F. Gringoli, and L. Salgarelli. Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting. *Computer Networks*, 2009.
42. M. Dusi, M. Crotti, F. Gringoli, and L. Salgarelli. Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting. *Computer Networks*, 2009.
43. A. Este, F. Gargiulo, F. Gringoli, L. Salgarelli, and C. Sansone. Pattern recognition approaches for classifying ip flows. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 885–895. Springer, 2008.
44. T. Fawcett. An introduction to ROC analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

45. G. Fedynyshyn, M. Ch. Chuah, and G. Tan. Detection and classification of different botnet c&c channels. In *Proceedings of the 8th International Conference on Autonomic and Trusted Computing*, 2011.
46. M. Feily, A. Shahrestani, and S. Ramadass. A survey of botnet and botnet detection. In *Proceedings of the 2009 Third International Conference on Emerging Security Information, Systems and Technologies*, 2009.
47. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, 1999. [Online].
48. P.-E. Forssén. Image analysis using soft histograms. In *Proceedings of the SSAB Symposium on Image Analysis: Norrköping*, 2001.
49. V. Franc, M. Sofka, and K. Bartos. Learning detector of malicious network traffic from weak labels. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2015.
50. E. P. Freire, A. Ziviani, and R. M. Salles. On metrics to distinguish skype flows from http traffic. In *2007 Latin American Network Operations and Management Symposium*, pages 57–66. IEEE, 2007.
51. J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*. Springer series in statistics Springer, Berlin, 2001.
52. K. Fukumizu, A. Gretton, X. Sun, and B. Schölkopf. Kernel measures of conditional dependence. In *NIPS*, 2008.
53. S. García, V. Uhlír, and M. Rehak. Identifying and modeling botnet c&c behaviors. In *Proceedings of the 1st International Workshop on Agents and CyberSecurity*, 2014.
54. S. García, A. Zunino, and M. Campo. Survey on network-based botnet detection methods. *Security and Communication Networks*, 2014.
55. Gartner. Market Guide for User and Entity Behavior Analytics. `https://www.gartner.com/doc/3134524/market-guide-user-entity-behavior`, 2015.
56. C. F. Gauss. Theoria interpolationis methodo nova tractata werke. *Göttingen: Königliche Gesellschaft der Wissenschaften*, 1886.
57. M. G. Genton. Classes of kernels for machine learning: A statistics perspective. *Journal of Machine Learning Research*, 2:299–312, 2002.
58. Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.
59. F. Giroire, J. Chandrashekar, N. Taft, E. Schooler, and D. Papagiannaki. Exploiting temporal persistence to detect covert botnet channels. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, 2009.
60. G. Gonzalez-Granadillo, S. Gonzalez-Zarzosa, and M. Faiella. Towards an enhanced security data analytic platform. In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications (ICETE 2018) - Volume 2: SECRYPT*, 2018.
61. A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 2012.
62. A. Gretton, D. Sejdinovic, H. Strathmann, S. Balakrishnan, M. Pontil, K. Fukumizu, and B. K. Sriperumbudur. Optimal kernel choice for large-scale two-sample tests. In *Advances in neural information processing systems*, pages 1205–1213, 2012.
63. M. Grill. *Combining network anomaly detectors*. PhD thesis, Czech Technical University in Prague, 2016.
64. M. Grill, T. Pevný, and M. Rehak. Reducing false positives of network anomaly detection by local adaptive multivariate smoothing. *Journal of Computer and System Sciences*, 2017.
65. M. Grill and M. Rehák. Malware detection using http user-agent discrepancy identification. In *2014 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 221–226. IEEE, 2014.
66. G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th Conference on Security Symposium*, 2008.
67. J. A. Gubner. *Probability and random processes for electrical and computer engineers*. Cambridge University Press, 2006.
68. J. A. Hartigan. *Clustering algorithms*. Wiley New York, 1975.

69. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer New York Inc., 2001.
70. D. M. Hawkins. *Identification of outliers*. Springer, 1980.
71. D. Herrmann, R. Wendolsky, and H. Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security*. ACM, 2009.
72. E. Hjelmvik and W. John. Statistical protocol identification with SPID: Preliminary results. In *Swedish National Computer Networking Workshop*, 2009.
73. R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys & Tutorials*, 16(4):2037–2064, 2014.
74. L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 1985.
75. M. Husák, M. Cermák, T. Jirsík, and P. ˇ Celeda. Network-based https client identification using ssl/tls fingerprinting. In *2015 10th International Conference on Availability, Reliability and Security*, pages 389–396. IEEE, 2015.
76. M. Jacomy, S. Heymann, T. Venturini, and M. Bastian. Forceatlas2, a continuous graph layout algorithm for handy network visualization. *Medialab center of research*, 2011.
77. J. T. Johnson. User behavioral analytics tools can thwart security attacks. `http://searchsecurity.techtarget.com/feature/User-behavioral-analytics-tools-can-thwart-security-attacks`, 2015.
78. K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 1972.
79. J. Jusko. *Graph-based Detection of Malicious Network Communities*. PhD thesis, Czech Technical University in Prague, 2017.
80. J. Jusko, M. Rehák, and T. Pevný. A memory efficient privacy preserving representation of connection graphs. In *Proceedings of the 1st International Workshop on Agents and CyberSecurity*, 2014.
81. J. Jusko, M. Rehak, J. Stiborek, J. Kohout, and T. Pevny. Using behavioral similarity for botnet command-and-control discovery. *IEEE Intelligent Systems*, 2016.
82. Ch.-H. Kao, J.-H. Dai, R. Ko, Y.-T. Kuang, Ch.-P. Lai, and Ch.-H. Mao. MITC Viz: Visual Analytics for Man-in-the-Cloud Threats Awareness. In *2016 International Computer Symposium (ICS)*, pages 306–311. IEEE, 2016.
83. T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: Multilevel traffic classification in the dark. *SIGCOMM Comput. Commun. Rev.*, 2005.
84. T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 1982.
85. M. Kopp, M. Grill, and J. Kohout. Community-based anomaly detection. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE, 2018.
86. M. Liberatore and B. N. Levine. Inferring the Source of Encrypted HTTP Connections. In *Proc. ACM conference on Computer and Communications Security (CCS)*, 2006.
87. S. Lloyd. Least squares quantization in PCM. *IEEE transactions on information theory*, 1982.
88. H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and Ch. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.
89. D. Lopez-Paz. *From Dependence to Causation*. PhD thesis, University of Cambridge, 2016.
90. Wei Lu, Yanyan Shen, Su Chen, and Beng Chin Ooi. Efficient processing of k nearest neighbor joins using mapreduce. *Proc. VLDB Endow.*, 5(10):1016–1027, June 2012.
91. A. Manukyan and E. Ceyhan. Classification of Imbalanced Data with a Geometric Digraph Family. *Journal of Machine Learning Research*, 2016.
92. R. Mehran, A. Oyama, and M. Shah. Abnormal crowd behavior detection using social force model. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 935–942. IEEE, 2009.
93. T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
94. H. D. K. Moonesinghe and P.-N. Tan. OutRank: a graph-based outlier detection framework using random walk. *International Journal on Artificial Intelligence Tools*, 2008.
95. E. H. Moore. On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society*, 1920.

96. K. Muandet, K. Fukumizu, F. Dinuzzo, and B. Schölkopf. Learning from Distributions via Support Measure Machines. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, 2012.
97. K. Muandet, K. Fukumizu, B. Sriperumbudur, B. Schölkopf, et al. Kernel mean embedding of distributions: A review and beyond. *Foundations and Trends in Machine Learning*, 10(1-2):1–141, 2017.
98. 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)*, pages 1–6. IEEE, 2017.
99. P.J. Nahin. *The Science of Radio*. AIP Press, 1996.
100. A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, 2001.
101. T. T. T. Nguyen and G. J. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys and Tutorials*, 2008.
102. David Novak, Michal Batko, and Pavel Zezula. Metric index: An efficient and scalable solution for precise and approximate similarity search. *Inf. Syst.*, 36(4):721–733, 2011.
103. David Novak, Martin Kyselak, and Pavel Zezula. On locality-sensitive indexing in generic metric spaces. In *Proceedings of the Third International Conference on SImilarity Search and APplications*, SISAP '10, pages 59–66, New York, NY, USA, 2010. ACM.
104. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.
105. R. Penrose. A generalized inverse for matrices. In *Mathematical proceedings of the Cambridge philosophical society*, 1955.
106. Tomáš Pevny and Andrew D Ker. Towards dependable steganalysis. In *IS&T/SPIE Electronic Imaging*, 2015.
107. J. Postel. User Datagram Protocol. RFC 768, 1980.
108. J. Postel. Internet Protocol. RFC 791, 1981.
109. J. Postel. Transmission Control Protocol. RFC 793, 1981.
110. G. Powell. *Beginning database design*. John Wiley & Sons, 2006.
111. C. E. Priebe, D. J. Marchette, J. G DeVinney, and D. A Socolinsky. Classification using class cover catch digraphs. *Journal of classification*, 2003.
112. J. G. Proakis and D. G. Manolakis. *Digital Signal Processing (3rd Ed.): Principles, Algorithms, and Applications*. Prentice-Hall, Inc., 1996.
113. A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, 2007.
114. W.M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 1971.
115. P. M. Roth, M. Hirzer, M. Köstinger, C. Beleznai, and H. Bischof. Mahalanobis distance learning for person re-identification. In *Person re-identification*, pages 247–267. Springer, 2014.
116. M. Roughan, S. Sen, O. Spatscheck, and N. Duffield. Class-of-service Mapping for QoS: A Statistical Signature-based Approach to IP Traffic Classification. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, 2004.
117. P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 1987.
118. W. Rudin. *Fourier Analysis on Groups*. Dover Books on Mathematics. Dover Publications, 2017.
119. G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 1975.
120. D. Schatzmann, W. Mühlbauer, T. Spyropoulos, and X. Dimitropoulos. Digging into HTTPS: Flow-based Classification of Webmail Traffic. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, 2010.
121. B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond (adaptive computation and machine learning)*. The MIT Press Cambridge, 2001.
122. M. Shashanka, M.-Y. Shen, and J. Wang. User and entity behavior analytics for enterprise security. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1867–1874. IEEE, 2016.

123. H. Shimazaki and S. Shinomoto. A method for selecting the bin size of a time histogram. *Neural computation*, 19(6):1503–1527, 2007.
124. E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, 2006.
125. L. Song. *Learning via Hilbert space embedding of distributions*. PhD thesis, University of Sydney, 2008.
126. K. Sricharan and A. O. Hero III. Efficient anomaly detection using bipartite k-NN graphs. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, 2011.
127. B. K. Sriperumbudur, A. Gretton, K. Fukumizu, B. Schölkopf, and G. R. G. Lanckriet. Hilbert space embeddings and metrics on probability measures. *Journal of Machine Learning Research*, 2010.
128. J. Stiborek, T. Pevný, and M. Rehák. Probabilistic analysis of dynamic malware traces. *Computers & Security*, 2018.
129. M. Subhransu and C. B. Alexander. Max-margin additive classifiers for detection. *2009 IEEE 12th International Conference on Computer Vision*, pages 40–47, 2009.
130. H. Sun, X. Sun, H. Wang, Y. Li, and X. Li. Automatic target detection in high-resolution remote sensing images using spatial sparse coding bag-of-words model. *IEEE Geoscience and Remote Sensing Letters*, 9(1):109–113, 2011.
131. D. J. Sutherland and J. Schneider. On the Error of Random Fourier Features. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, pages 862–871. AUAI Press, 2015.
132. Cisco Systems. Cisco Annual Security Reports. `https://www.cisco.com/c/en/us/products/security/security-reports.html`, 2019.
133. A. S. Tanenbaum and D. J. Wetherall. *Computer Networks*. Prentice Hall Press, 5th edition, 2010.
134. F. Tegeler, X. Fu, G. Vigna, and Ch. Kruegel. Botfinder: Finding bots in network traffic without deep packet inspection. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, 2012.
135. Ch.-F. Tsai. Bag-of-words representation in image annotation: A review. *ISRN Artificial Intelligence*, 2012.
136. V. Valeros, P. Somol, M. Rehak, and M. Grill. Cognitive Threat Analytics: Turn Your Proxy Into Security Device. `https://blogs.cisco.com/security/cognitive-threat-analytics-turn-your-proxy-into-security-device`, 2016.
137. L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.
138. A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Transactions Pattern Anal. Mach. Intell.*, 2012.
139. A. Vedaldi and A. Zisserman. Sparse kernel approximations for efficient classification and detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.
140. P. Velan, M. Čermák, P. Čeleda, and M. Drašar. A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management*, 2015.
141. N. V. Verde, G. Ateniese, E. Gabrielli, L. V. Mancini, and A. Spognardi. No NAT'd User Left Behind: Fingerprinting Users behind NAT from NetFlow Records Alone. In *Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on*, 2014.
142. M. P. Wand. Data-based choice of histogram bin width. *The American Statistician*, 51(1):59–64, 1997.
143. K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 2009.
144. Ch. K. I. Williams and M. Seeger. Using the Nyström Method to Speed Up Kernel Machines. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, 2000.
145. C.V. Wright, F. Monrose, and G. M. Masson. On inferring application protocol behaviors in encrypted network traffic. *Journal of Machine Learning Research*, 2006.
146. R. Xu and D. C. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 2005.

147. W. Zaremba, A. Gretton, and M. Blaschko. B-tests: Low Variance Kernel Two-sample Tests. In *Advances in Neural Information Processing Systems 26*, 2013.
148. Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search: The Metric Space Approach*. Springer, 2005.
149. E. Zhang and M. Mayo. Improving bag-of-words model with spatial information. In *2010 25th International Conference of Image and Vision Computing New Zealand*, pages 1–8. IEEE, 2010.
150. D. Zhao, I. Traore, B. Sayed, W. Lu, S. Saad, A. Ghorbani, and D. Garant. Botnet detection based on traffic behavior analysis and flow intervals. *Computers and Security*, 2013.
151. J. Zhao and D. Meng. Fastmmd: Ensemble of circular discrepancy for efficient two-sample test. *Neural Computation*, 27:1345–1372, 2015.
152. R. Zhao and K. Mao. Fuzzy bag-of-words model for document representation. *IEEE Transactions on Fuzzy Systems*, 26(2):794–804, 2017.

Appendix

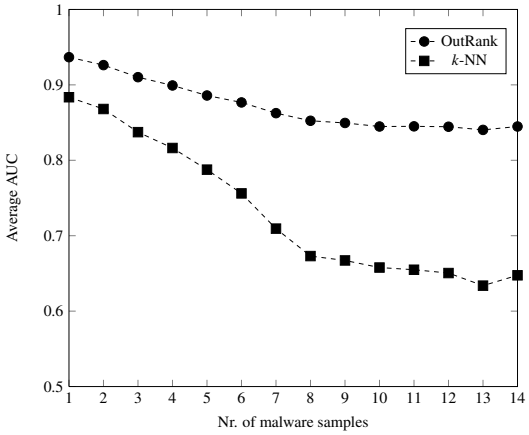
List of application names in the *TCP flows* dataset

Full list of application names:

```
'APSDaemon.exe', 'AddressBookSourceSync', 'AgentService.exe', 'App Store', 'AppleIEDAV.exe',
'Arellia.Agent.Service.exe', 'Box Sync', 'CalendarAgent', 'CcmExec.exe', 'CiscoJabber.exe',
'Cisco Jabber', 'Cisco.WebEx.Start', 'Dropbox109', 'DropboxOriginal', 'Dropbox.exe', 'Evernote',
'FireSvc.exe', 'GoogleSoftwareUpdateAgent', 'GoogleUpdate.exe', 'Google Chrome', 'Mail',
'Meeting Center', 'Microsoft Outlook', 'OUTLOOK.EXE', 'Python', 'Safari', 'SkyDrive.exe',
'Skype', 'SoftwareUpdateCheck', 'Spotify', 'SubmitDiagInfo', 'VpxClient.exe', 'WebProcess',
'WmiPrvSe.exe', 'apsd', 'atmgr.exe', 'chrome.exe', 'cma', 'com.apple.WebKit.Networking',
'com.apple.WebKit.WebContent', 'com.apple.iCloudHelper', 'cscan.exe', 'firefox-bin', 'firefox.exe',
'fpsaud', 'gconsync', 'googledrivesync.exe', 'helpd', 'iCloudServices.exe', 'iTunes',
'iTunes.exe', 'iexplore.exe', 'jamf', 'java', 'ksfetch', 'lsass.exe', 'mdmclient', 'mutt',
'netsession_win.exe', 'pcdrcui.exe', 'sfc.exe', 'softwareupdate', 'softwareupdated', 'splwow64.exe',
'ssh', 'storeagent', 'taskhost.exe', 'thunderbird.exe', 'vmnat.exe', 'vmware-vmrc.exe'
```

OutRank stability

In Chapter 5, we have argued that OutRank algorithm should be more robust with respect to small clusters of outliers which we can expect to be present in real data. Figure below shows average AUC of the OutRank and k-NN algorithms with representations based on soft histograms for varying number of malware samples inserted into the background data sets in the outlier detection experiment. The average is calculated over all three background networks and random selections of malware. We can observe that the AUC of the k-NN detector drops more rapidly than that of OutRank as the number of infected users increases. This means that in our outlier detection settings the OutRank is indeed more robust against multiple concurrent infections.



Average values of AUC for OutRank and k-NN outlier detection algorithms with fingerprints based on soft histograms for varying number of malware samples inserted into the background data sets. The average is calculated over all three background data sets and random selections of malware.

Approximate similarity search pseudocode

Below we present listing of the pseudocode which illustrates implementation of the approximate similarity join in MapReduce environment described in Chapter 6:

Replication and approximate similarity search

```
1: map-setup. //get IDs of groups to which each pivot's cell should be replicated
2:     nearestGroups = ComputeNearestGroups().

3: map (k1, v1)
4:    if k1.dataset == ℚ then    //query object
5:        groupID = GetGroupID(k1.cell)
6:        emit(groupID, (k1, v1))
7:    else                       //reference object
8:        pivotID = GetPivotID(k1.pivot)
9:        foreach groupID in nearestGroups[pivotID] do //replication of reference objects
10:           emit(groupID, (k1, v1))
11:       end
12:   endif

13: reduce (k2, v2)
14:    parse objects from ℚ into D_ℚ and from 𝕊 into list L of Voronoi cells {ℂ_i^𝕊}
15:    foreach q in D_ℚ do
16:       compute distance to pivots δ(q, p_i) and sort Voronoi cells in L
17:       kNN = ∅              //k-NN result
18:       r = MAX_VALUE   //query radius
19:       foreach ℂ_i^𝕊 in L do  // for each Voronoi cell check its objects
20:          if δ(q, p_i) > ℂ_i^𝕊.r_i + r then continue      //query-cell overlap check
21:          foreach o_𝕊 in ℂ_i^𝕊 do
22:             if |δ(q, p_i) − δ(o_𝕊, p_i)| > r then continue    // lower bound filter
23:             distance = δ(q, o_S)
24:             if distance ≥ r then continue
25:             update kNN by o_𝕊
26:             r = δ(q, kNN[k])    //radius = distance from q to k-th object o_𝕊
27:          end
28:       end
29:       output(q, kNN)
30:    end
```

ComputeNearestGroups()

```
1: nearestGroups = array of size equal to the number of pivots
2: foreach p_i in ℙ do
3:     dist_{p_i} = ∅                //empty set of distances to all other pivots
4:     foreach p_j in ℙ do    //for each pivot combination
5:         groupID = GetGroupID(p_j) //group id where p_j belongs
6:         dist = δ(p_i, p_j)            //distance between pivots
7:         add pair <groupId; dist> to dist_{p_i}
8:     end
9:     sort dist_{p_i} in ascending order by dist in pairs
10:    nGs = ∅
11:    pivCount = 0      //number of considered nearest pivots - for approximation
12:    foreach <groupId; dist> in dist_{p_i} do
13:        if not nGs contains groupId then
14:            add groupId to nGs
15:        endif
16:        pivCount++
17:        if pivCount > t_r then break. //replication threshold check
18:    end
19:    pivotID = GetPivotID(p_i)
20:    nearestGroups[pivotID] = nGs
21: end
22: return nearestGroups
```

FP-50 error measure

Formal definition of the FP-50 error measure from [106] used in Chapter 6:

$$error = \frac{1}{|\mathbb{N}|} \sum_{x \in \mathbb{N}} \left[f(x) > \text{median}\{f(y) | y \in \mathbb{P}\} \right],$$

where $\mathbb{N}$ is a set of negative (benign) objects, $\mathbb{P}$ is a set of positive (malicious) objects and $f(x)$ is a classification score for an object x.

The ECM classifier assigns classification score as:

$$f(x) = w^{\mathrm{T}} x$$

The weights w are trained by solving the following optimization problem:

$$w^* = \underset{w}{\text{argmin}} \sum_{x \in \mathbb{N}} \exp\{(x - \mu)^{\mathrm{T}} w\},$$

where μ is a mean of objects from the positive class $\mathbb{P}$.

Application of AMRep in Support Measure Machines

Support Measure Machines (SMM) [96] were proposed as an extension of Support Vector Machines (SVM) to spaces of probability distributions, where the classified objects are not single points, but probability distributions. Similarly as kernel functions are utilized in standard SVM, kernel functions on top of probability probability distributions are leveraged in SMM, as introduced in Chapter 7 (Section 7.2). Specifically, if the Gaussian kernel $\exp(-\gamma \|\mu_p - \mu_Q\|^2_{\mathcal{H}})$ is used, it uses the distances $\|\mu_P - \mu_Q\|_{\mathcal{H}}$ that correspond to $\text{MMD}(P, Q)$, as discussed in Chapter 7. The Gaussian kernel on distributions can be therefore rewritten as $\exp(-\gamma \, \text{MMD}^2(P, Q))$.

However, a limiting factor for practical usage of SMM on larger problems is their computational complexity. By considering the approximate computation of $\text{MMD}^2(P, Q)$ as introduced in Section 7.4 of Chapter 7, the Gaussian kernel can be approximated as

$$\exp(-\gamma\|\mu_P - \mu_Q\|^2_{\mathcal{H}}) \approx \exp(-\gamma\|\mathbf{L}^{-1}\left(\varphi(\mu_P) - \varphi(\mu_Q)\right)\|^2),$$

with the meaning of $\mathbf{L}$ and φ as introduced in Chapter 7. This way we obtain an approximate version of SMM, which is in fact a conventional SVM with Gaussian kernel working on top of AMRep representations of the distributions.

In order to compare accuracy and computational requirements of this approximate version of SMM with the original SMM, we created a dataset which reflects parameters of the "Synthetic dataset" used in experiments of [96]. The only difference is that the authors of [96] worked with distributions specified by their parameters, not with empirical samples from that distributions. The reason why we did not use the parametric specifications of the distributions is that our framework for MMD approximation is proposed for the situations in which the parameters of the underlying distributions are not known and only the samples are available. We believe that this reflects most practical situations when information about the distributions is obtained by collecting the empirical samples from them.

Results of this comparison are presented below[1]. The experiment compared accuracies and classification times depending on the number of samples (i.e., sample sets that contained the empirical observations) and size of each sample (i.e., number of observations in one sample set). We used LibSVM [22] implementation of the SVM classifier for the approximate version. As we can see, the approximate version of SMM which uses AMRep is able to significantly speed-up the classification with only a slightly decreased accuracy. Therefore, considering the AMRep for approximate versions of kernel-based classifiers working on top of probability distributions is a perspective line of future research.

Sample set size	SMM orig.	SVM+AMRep
5	90.8% (0.61s)	90.5% (0.05s)
15	96.4% (3.47s)	95.9% (0.10s)
30	96.8% (12.72s)	96.7% (0.21s)
45	98.0% (33.43s)	97.5% (0.29s)
60	98.5% (53.58s)	97.6% (0.39s)

Accuracies (%) and classification times (in seconds) depending on the number of observations in one sample set. Training dataset always contained 1600 samples, testing dataset always contained 400 samples (positive:negative ratio was 1:1).

Number of sample sets	SMM orig.	SVM+AMRep
1000 (200)	92.9% (1.29s)	93.0% (0.03s)
1200 (240)	93.3% (1.82s)	93.0% (0.03s)
1500 (300)	93.5% (2.48s)	93.8% (0.04s)
2000 (400)	94.2% (5.06s)	94.0% (0.07s)
4000 (800)	crashed	94.3% (0.27s)
10000 (2000)	crashed	94.4% (1.34s)

Accuracies (%) and classification times (in seconds) depending on the number of sample sets (total and in the testing set, testing set size in parenthesis). Each sample set contained 10 observations.

[1] Implementation of SMM was obtained from the page `http://webdav.tuebingen.mpg.de/smm/`