INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Near real-time network analysis for the identification of malicious activity

Rafael Cardoso de Oliveira - a35096

Dissertation presented to the School of Technology and Management of Bragança to obtain a Master's Degree in Informatics.

Supervised by:
Prof. Tiago Miguel Ferreira Guimarães Pedrosa
Prof. Rui Pedro Sanches de Castro Lopes

This dissertation does not include the criticisms and suggestions made by the Jury.

Bragança
2020-2021

Near real-time network analysis for the identification of malicious activity

Rafael Cardoso de Oliveira - a35096

Dissertation presented to the School of Technology and Management of Bragança to obtain a Master's Degree in Informatics.

Supervised by:

Prof. Tiago Miguel Ferreira Guimarães Pedrosa

Prof. Rui Pedro Sanches de Castro Lopes

This dissertation does not include the criticisms and suggestions made by the Jury.

Bragança

2020-2021

Dedication

To my family.

Acknowledgements

I would like to thank all the professors from the Institute Polytechnic of Bragança for all the help throughout this academic journey.

Special thanks to my supervisors, Prof. Tiago Pedrosa and Prof. Rui Lopes for giving invaluable help, without it, my research would have been impossible. Thank you.

For last but not least I would like to thank my family for all the support.

Resumo

A evolução da tecnologia e o aumento da conectividade entre dispositivos, levam a um aumento do risco de ciberataques. Os sistemas de deteção de intrusão são essenciais para tentar prevenir, detetar e conter a maioria dos ataques. No entanto, o aumento da criatividade e do tipo de ataques aumenta a necessidade dos sistemas de proteção possuírem cada vez mais recursos e poder computacional. Por sua vez, requerem escalabilidade horizontal para acompanhar a massiva infraestrutura de rede das empresas e a complexidade dos ataques. Tecnologias como machine learning apresentam resultados promissores e podem ser de grande valor na deteção e prevenção de ataques em tempo útil. No entanto, a utilização dos algoritmos e ferramentas requer sempre um conjunto de dados sólidos e confiáveis para treinar os sistemas de proteção de maneira eficaz. A implementação de um bom conjunto de dados requer sistemas horizontalmente escaláveis, robustos, modulares e tolerantes a falhas para que a análise seja rápida e rigorosa. Este trabalho descreve a arquitetura de um sistema de captura, armazenamento e análise, capaz de capturar pacotes de múltiplas fontes e analisá-los de forma paralela. O sistema depende de vários nós modulares com funções específicas para oferecer suporte a diferentes algoritmos e ferramentas.

Palavras-chave: Cibersegurança, IDS, Sistemas Distribuídos, Machine-Learning

Abstract

The evolution of technology and the increasing connectivity between devices lead to an increased risk of cyberattacks. Reliable protection systems, such as Intrusion Detection System (IDS) and Intrusion Prevention System (IPS), are essential to try to prevent, detect and counter most of the attacks. However, the increased creativity and type of attacks raise the need for more resources and processing power for the protection systems which, in turn, requires horizontal scalability to keep up with the massive companies' network infrastructure and with the complexity of attacks. Technologies like machine learning, show promising results and can be of added value in the detection and prevention of attacks in near real-time. But good algorithms and tools are not enough. They require reliable and solid datasets to be able to effectively train the protection systems. The development of a good dataset requires horizontal-scalable, robust, modular and fault-tolerant systems so that the analysis may be done in near real-time. This work describes an architecture design for horizontal-scaling capture, storage and analyses, able to collect packets from multiple sources and analyse them in a parallel fashion. The system depends on multiple modular nodes with specific roles to support different algorithms and tools.

Keywords: Cybersecurity, IDS, Distributed-Systems, Machine-Learning

Contents

List of Tables

List of Figures

Listings

Acronyms

APT Advanced Persistent Threats. xix, 8

ARP Address Resolution Protocol. xix, 32, 47

C&C Command and Control. xix, 16, 17

CA Certificate authority. xix, 37

CNN Convolutional Neural Network. xix, 52

DAG Data Acquisition and Generation. xix, 21

DDoS Distributed Denial of Service. xix, 11–13, 15

DFI Deep Flow Inspection. xix, 13, 14

DGA Domain Generation Algorithms. xix, 34, 35, 68

DHCP Dynamic Host Configuration Protocol. xix, 37

DNS Domain Name System. xix, 16, 17, 48

DoS Denial of Service. xix, 13, 14

DPI Deep Packet Inspection. xix, 13, 14

FIFO First In First Out. xix, 29

HDFS Hadoop Distributed File System. xix, 8, 11, 12, 26–28, 34, 46, 60

ICMP Internet Control Message Protocol. xix, 47

ID2T Intrusion Detection Dataset Toolkit. xix, 13

IDS Intrusion Detection System. viii, xix, 6, 9, 11, 17, 18

IoT Internet of Things. xix, 17, 28, 72

IP Internet Protocol. xix, 11, 12, 14–16, 20, 47, 72

IPS Intrusion Prevention System. viii, xix, 6, 72

IT Information Technology. xix, 1, 2, 33, 71

JVM Java Virtual Machine. xix, 28

KiB Kibibytes. xix, 44, 56, 57, 59, 60

KPIs Key Performance Indicators. xix, 7

LSTM Long short-term memory. xix, 50, 52

MB Megabyte. xix, 44

NAT Network Address Translation. xix, 32

NIC Network Card Interface. xix, 21, 37–39, 41

NS Name Server. xix, 16

P2P Peer To Peer. xix, 13

PPA Privacy Policy Agreement. xix, 17

QoS Quality of Service. xix, 7, 8, 17

RAM Random access memory. xix, 38–41

RNN Recurrent Neural Network. xix

SDN Software Defined Network. xix, 16

SSD Solid State Drive. xix, 38, 39

SSL Secure Sockets Layer. xix, 37, 43

TCP Transmission Control Protocol. xix, 47

TLD Top-level domain. xix, 48, 49, 52

TLS Transport Layer Security. xix

UDP User Datagram Protocol. xix, 47

Glossary

Distributed system A form of computing in which data and applications are distributed among disparate computers or systems, but are connected and integrated by means of network services and interoperability standards such that they function as a single environment [1].

Modular system A design principle that subdivides a system into smaller parts called modules, which can be independently created, modified, replaced, or exchanged with other modules or between different systems offering flexibility and variety in use [2].

Promiscuous mode In promiscuous mode, a network adapter does not filter packets. Each network packet on the network segment is directly passed to the operating system or any monitoring application [3].

Scalability Measure of a system ability to increase or decrease in performance and cost in response to changes in application and system processing demands [4].

Chapter 1

Introduction

Cybersecurity attacks have been an issue for many years and, as time passes, these attacks get more sophisticated, increasing the difficulty to detect and prevent them. Any size companies must have a solid system to prevent those attacks, as it certainly will have devastating consequences in case of a breach. Intellectual property and personal data are two kinds of information that have considerable value [5], meaning that some people will try to get them and profit from them. Cyberattacks' objective is not only to steal companies' information, but also to deny services that companies provide. In any case, a successful cyberattack on any company will most likely make them lose capital, opportunities or even force them to close their business.

It is foreseen that, by the end of 2025, a total of around 8,000 million people and 41,200 million devices are connected to the internet, with 10,300 million of them not being IoT devices (laptops, desktops, smartphones, etc.) [6], [7]. With this huge number of devices connected to the Internet, the companies that provide online services (social media, banking, retail, cloud, etc.), will also need to expand to be able to keep up with the demand. This growth will consist in an expansion of the companies' Information Technology (IT) infrastructures, adding more servers and routing devices. This increase will also result in a higher probability of suffering a cyberattack, eventually with the whole IT infrastructure being compromised.

Cyberattacks are the main problem of the digital world and, according to Lysenko et al. [8], they have generated financial damage of around 1,500,000 million U.S. dollars in 2019. Small companies are the most fragile since around 60% of them close within six months of an incident [9]. But that does not mean that medium to large companies are safe from serious trouble since leaked intellectual property or stolen user data can have a severe negative impact on any company. In June of 2021, the data of over 700 million Linkedin users (about 92%) got exposed. The leaked data consists of the following personal details: phone numbers, physical addresses, geolocation data and more [10]. CD Projekt, in February of 2021, got breached and all of the data stolen (accounting, administration, HR) including the source code of multiple projects (that were sold later by the perpetrators) not to mention the hours that the employees were unable to work, costing the company even more funds [11]. Yahoo, in 2016, announced that in 2013/2014 they suffered a security breach compromising 3,000 million user accounts [12], including real names, email addresses, dates of birth and telephone numbers [13]. At the time Yahoo was being acquired by another company and after this announcement they lower the offer by 350 million U.S. dollars.

This is a problem that the community must face to prevent further successful attacks on corporations, that are the primary entity to suffer the consequences. To stop this kind of incidents a company must possess a system capable to analyse the network traffic in near real-time. The number of cyberattacks is growing both in number and complexity, there are always new ways of breaching and compromising the networks and, with this complexity, traditional systems are not effective anymore due to the lack of successful detection and prevention of attacks and the ability to operate with complex IT infrastructure.

1.1 Problem statement

Some recent systems lack the ability to analyse the network data in near real-time, they just perform the analysis pos-capture, making the system only useful to detect if a breach happened, and not to prevent one. The challenge is to design and implement a system capable of analysing the network traffic in near real-time, so it can detect and prevent cyberattacks as they occur.

1.2 Objectives

This dissertation aims to study and develop a solution to capture, analyze and process network traffic in near real-time, with the objective of detecting anomalies, such as possible cyberattacks and/or the presence of malicious programs that affect the proper functioning of the network. A literature review will be conducted in order to technologically and scientifically support the work to be developed and to serve as a basis for the design of a reliable, modular and scalable architecture for the capture, storage and analysis of network traffic. The analysis of traffic will be performed by developing solutions that, through the behaviour of the network, will highlight situations of potentially malicious activity on the network. On figure 1.1 it's presented this dissertation work schedule.

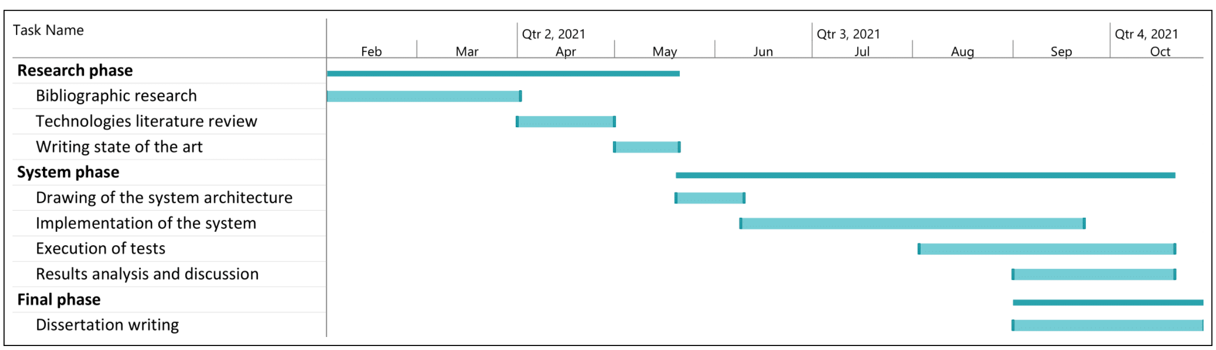

Figure 1.1: Dissertation work schedule

1.3 Document structure

This document is structured in six chapters. Chapter 1 refers to the introduction, the problem statement description, the objectives of this dissertation, the document structure and the contributions made throughout this study. Chapter 2 explains the research approach taken to gather vital information, the literature review and the tools used on the proposed system as well the proper justification. Chapter 3 explains the approach taken to solve the identified problem. Chapter 4 explain the implementation of this work. Chapter 5 presents every experiment performed to evaluate and improve the proposed system. Lastly, chapter 6 layout conclusions and future work.

1.4 Contributions

During the execution of this work, the following contributions were made:

- Publication of the following paper (appendix B): Oliveira, R. et al. (2021) "A scalable, real-time packet capturing solution" in International Conference on Optimization, Learning Algorithms and Applications. Springer.
- A Python script that plots iPerf3's JSON file, available at `https://github.com/rafaeloliveira00/iperf3-plotter`

1.5 Acknowledgments

This work was developed within the "CybersSEC IP - CYBERSecurity SciEntific Competences and Innovation Potential (NORTE-01-0145-FEDER-000044)" research project.

Chapter 2

Background

This chapter is arranged into seven sections, the first explains the methodological approach used in the literature review. The next provides a discussion of a set of related works giving a better understanding of the problems that the scientific community has identified, possible solutions and the problems of these solutions. Then other application scenarios where network data may be used, the functional and security requirements, the different network data tiers, the layout of the PCAP file format and for the last section a set of tools and technology used in the proposed system with the proper justification on why they got selected.

2.1 Literature review methodology

This section includes the research methodology of this work. It details the research strategy, the methodology approach, the methods of data collection and analysis, the tools used to find and organize the information and the justification of methodological choices.

To solve the problems described in section 1.1, research was performed to acquire a better understanding of the topics and also the current problems and solutions. This way, knowing what the research community is doing, the problems they face and the improvements that could be done, there is a much better chance to create a solid system and produce good results. This also serves the purpose to describe the characteristics of some concepts like IDS, IPS and others.

For some topics like machine learning algorithms, secondary data gathering (results of the experiments like, time of executions, accuracy, etc.) was necessary to understand which algorithms offer the best accuracy in finding network security threads.

The research began in searching for surveys about network data collection and analysis of the last four years. With those surveys, it was possible to know many important concepts and what the research community was doing to capture, store and analyse network traffic in a modern networking infrastructure with high bandwidth. Moreover, it also provided insights to the trend technologies to solve these issues and which functional and security requirements a solid system must have to work without flaws. This way it was easier to find and filter other researches that could help.

As surveys are also a collection of works, they contain multiple citations of other researches that contains valuable information.

2.1.1 Data collection

The Online Knowledge Library b-on (`www.b-on.pt`) was the primary search engine to find research papers, surveys, magazines, etc. This search engine includes multiple well known digital libraries such as IEEE (`ieeexplore.ieee.org`), Elsevier (`www.elsevier.com`), Springer (`https://link.springer.com`), and many others.

Multiple keywords were used in the search bar, some alone and others joined with boolean operations: packet capture, packet storage, packet analysis, neural network, intrusion detection, passive DNS, big data analysis, distributed file system, distribution processing, network forensics, stream processing, dga. In conjunction with the keywords at the start, the search was also filtered by the age of the publication, up to four years.

2.1.2 Method of analysis

After acquiring a set of articles, the next step is to check if they go accordingly to the problem statement. The approach was to read the abstract and check the paper' relevance and use it to further analyse or discard it.

The next step to do when an article is accepted to further read and analysis is to take notes about that work, extracting the problem that the authors identified, what was their approach to fixing the problem, with which technologies, the design of the system architecture, the parameters they have used in their experiments, the final results and problems that were encountered. This way, with this knowledge, the best technology with the proper parameters may be used in this research, avoiding making the same flaws as others and avoiding making work that is already done. Each article also provides a source for further references were more information could be found.

2.2 Literature review

P. Roquero et al. [14] attempt to solve Quality of Service (QoS) problems on the network while they're capturing network packets. They present a scalable data collection that captures the packets at multiple points of the network and sends them to multiple receivers that will perform the analysis. In their system design, they have the main entity that controls the whole system. That entity, designated by the orchestrator, keeps the states of every component, it commands the microsniffers when to start and stop the capture, which filters to use while capturing the data and to which devices (designated by monitoring sinks) it must send the captured data. The microsniffers also send to the monitoring sinks a set of Key Performance Indicators (KPIs) so they may identify if the QoS is in the desired level. The network KPIs that the authors selected are the following: loss of connectivity, micro-saturation, packet loss, congestion of user terminal or server, latency and security issues (access to malicious servers). The microsniffers are codded in C++ and the whole system' communications are encrypted with SSL; also, authentication is performed between the orchestrator and sinks. Besides offering a good approach to perform packet capture and at the same time monitor the QoS of the network, this system possesses a huge flaw in its design: data capture is performed by a swarm of software probes, that have to be installed in all the network computers; this requires access to each computer and individual installation and configuration; also it would be difficult to have the authorization to install the software in sensitive servers, not to talk about systems where it would be just impossible to install this kind of software, like IoT devices, embedded systems, and others alike.

Nowadays, Botnets are among the most serious network security threads, especially Advanced Persistent Threats (APT) that may take around two years to infect hosts. S. Mousavi et al. [15] identify the main problems related to the detection of botnets: traditional Botnet detection methods have trouble to scale to meet the needs of multi-Gbps networks and most published detection solutions are not effective on networks with millions of users. To solve this problems they propose a framework fully scalable to successfully detect Botnets on networks. Their framework is split into the following five scalable modules: network traffic processing, transportation of the data, processing of the data, storage and analysis. Their detection strategy is using real network data as the data source, standard traffic monitoring as the data type, anomaly-based as the classification method and passive analyses as the interaction with Botnet. On the network traffic processor module, traffic mirroring is used to extract the information from the routing devices and send it to the probe that is using PF_Ring. After the probe has captured the data, it will send the data to a scalable queue (scalable queuing module). That module, supported by Apache Kafka, is responsible for only synchronizing the data producers (probes) and receivers and aggregate the data in one queue (data processing of any kind is not performed in this module). The stream processing module, supported by Apache Spark Streaming, is responsible to gather the data from the queue and storing it on the storage module, built on the Hadoop Distributed File System (HDFS). The processing module also extracts some features that will support the machine learning algorithms in the detection of Botnets. The analysis module is then responsible to receive the extracted features and using them on algorithms and tools like Apache SparkML to help with the detection of Botnets. With their framework, they can operate in a 5Gb/s bandwidth network. This framework is well designed and allows full scalability on every module. However, PF_Ring requires a paid licence to be used.

P. Emmerich et al. [16] defend that capturing only samples of the network traffic is not enough to detect the origin of the attacks, so should be captured the full packet. They also state that it's hard to find a solution that can keep up with the transfer rate of modern networks. In order to fix these issues, they proposed a tool, designated by FlowScop, with the capability of operating at rates of 100 Gbit/s and 120 million packets per second. They accomplish these results by developing a queuing data structure (queue of queues) and by trading latency for throughput. They use an in-memory buffer as intermediate storage, to store the network packets and only dump them to the disk if the filter matches. The problem with their solution is that, it is centralized, after the packets are captured they are written to a local disk, making it impossible to do the analysis in real-time.

Arkime is an open-source, fully scalable, full packet capture, indexing and database system (Elasticsearch). It exposes APIs which allows PCAP and JSON data to be downloaded and consumed by other services like an IDS. It stores and exports all packets in standard PCAP format. Since Arkime is only used to capture, store and share the information, J. Uramová et al. [17] mission is to find a good IDS that works well with Arkime. The first experiment was with the integration of Snort IDS. To make that integration between these two, they've used an available plugin (Pigsty) but that functionality does not work anymore since its support was ceased and a new tool was created. But the new tool only supports the Suricata IDS; that's the reason why the authors stopped with the integration between Arkime and Snort and will try on future work to integrate Arkime with Suricata IDS. In their work, they also present a set of formulas that help to know how many nodes are required to store packets, depending on the network bandwidth and the number of days whose data will be stored; they also point out that, as in Arkime, is very easy to add nodes, being better to start at a lower number and increase as needed.

M. Saavedra et al. [18] recognises that network traffic is still being analyzed on vertically scalable machines and defend that Hadoop's horizontal scale ecosystem provides a better environment for processing captured network packets stored in PCAP files. PCAP file format wasn't designed for heavy processing so they point out that there isn't a straightforward method to analyze this kind of file on Hadoop. Their main contribution is an improvement of an existing framework for Hadoop that is capable of processing the PCAP files without the need in converting them into a Hadoop supported data structure like Apache Parquet. They've run some tests to compare the processing time of a query (the total size of the PCAP is omitted by the authors) on the PCAP file using the existing framework, their improvement on that framework and on the Apache Parquet dataset that resulted from the conversion of the PCAP file. They've performed the analysis on a host with intel i5-2500@3.3GHz and 4GB of RAM. The preprocessing was done on a similar host but with 16GB of RAM instead. On 90 minutes of captured data, the authors' improvement and the Apache Parquet format took about 14, 9 and 1 hour respectively of query execution time. The authors managed an improvement of the existing framework but their approach is far from the Apache Parquet results. Or not; it all depends on the situation since the conversion of the PCAP file into Apache Parquet took about 21 hours using hcap and some information about the packet may be lost after conversion.

M. Tun et al. [19] correlate the increase of cyberattacks with the enhancement of the network traffic. With this huge amount of stream data flows within a short period, the successful detection of cyberattacks in real-time is a challenge. To fix this issue the authors defend that big data streaming analysis can achieve real-time using Apache Kafka and Apache Spark Streaming. They defend it's among the best software approach based on the following reasons: Apache Spark is up to 100x faster than Hadoop's Map Reduce; Spark Streaming recovers from node failure without losses; there is no need to worry about duplication as it offers exactly-once delivery; easy to use; highly scalable; low latency and fault tolerance. The purpose of their work is to investigate the impact of processing time on the number of stream records and to improve the efficiency of Apache Kafka and Apache Spark Streaming. They've performed their experiment on an Intel i5 processor machine with 8GB of RAM on Linux Ubuntu 18.04 LTS. The streaming software versions are the following: JDK 11, Scala 2.11.12, Apache Kafka 2.12 and Apache Spark 2.1.0. They've used 500,000 records (total of 120MB) of the UNSW-NB15 dataset that provides 100GB of network traffic on a PCAP file. To clarify, batch interval tells Apache Spark Streaming the time interval to fetch the data in the Kafka cluster. After multiple executions with different batch intervals, they conclude that the best batch interval is between 30 and 50 seconds, which took less than 1 second of processing time. The worst batch interval was 1 second that took about 6 seconds of processing time. To note that a higher batch interval may remove the capability to process the network data in real-time.

A. Karimi et al. [20] reinforce the idea that companies need to have proper attack detection and mitigation tools. With the increase of the traffic volume, feature extraction for machine learning algorithms will be computational exhausting, imposing several challenges in the implementation of a real-time IDS. Even with high-end stand-alone systems, extraction of features takes a lot of time even on offline analysis. To address these issues the authors propose a system architecture of an IDS that could run in near real-time to detect different kinds of Distributed Denial of Service (DDoS) attacks. They use Netmap in conjunction with port mirroring to perform the network packet capture and Apache Spark to compute the data and extract features. They also use Apache Hadoop on the same cluster as Apache Spark; this way they can use HDFS for distributed storage. Their system performs five different tasks: the first is the live capturing and extraction of requirement headers; the second is the distribution of the extracted headers throughout the storage cluster; the third is the extraction of features from the packet headers; the fourth is the analysis of the traffic features for the anomaly detection; the last task consists in the training and update of the machine learning algorithm using Apache Spark Mlib. The last task is only performed in a big-time window like, for example, each month after the first training. They use the following features in their machine learning model: source Internet Protocol (IP), destination IP, source port, destination port, IP protocol, IP payload, TCP FIN flag, TCP SYN flag, TCP RST flag, TCP ACK flag, TCP RUG flag and TCP PUSH flag. In their experiments, they've used a dataset containing 40 minutes of a DDoS attack that generated a PCAP file of 21.1 GB. For the Spark cluster and HDFS they used a server with two Intel Xeon CPUs at 2.30 GHz, 96GB of RAM and 20 cores per CPU running VMware ESXi. With six worker nodes, a 3 GB file took around 3.17 minutes to get processed, while it took 3.5 minutes to get processed with four workers. The authors leave a note saying that the feature extraction could be faster if they executed some queries in parallel.

F. Aryeh et al. [21] identify that plenty of institutions' information is accessed illegally and the Wireshark, a generally used tool, is most of the time impractical to filter and follow TCP streams. To solve this problem they developed a system, consisting of the capture, storage and analysis of network packets. In the capturing process, they use Scapy, a Python framework, to capture the packets and store them in a PCAP file format. They justified the choice in using Scapy as it has ongoing Python community support and many capabilities. The PCAP files are converted into a Pandas (Python framework) DataFrame in order to do the analysis. When the data is converted, it is possible to retrieve some information and generate graphics about the top 1 source address, the top 1 destination address, to whom the top 1 address communicates too, the most used ports, etc. They've captured network traffic for about 2 hours generating only 86,000 packets and with the generated graphs they were able to retrieve a suspicious IP address. The authors omitted the bandwidth of the network, the computation power and the time taken in the conversion and queries of the data. Clearly, the authors didn't want a system able to capture on a multi-Gb network, since Scapy is not able to capture at such high rates. Moreover, as their system is able only to analyse the data after the capture, it can't fix the problem that they identified, that is preventing the unauthorized access to the companies data (they can only identify suspicious hosts).

E. Do et al. [22] identify that detecting and labelling network attacks is yet a huge challenge. They outline a machine learning algorithm technique that uses deep neural networks to detect and classify a diversity of network attacks. As they use PCAP files as their data source they first perform a pre-processing of the raw data. The PCAP file data is transformed from packets to flows using the YAF and YAFSCII tools. Those tools convert the data into a human-readable format. Then they use Shannon entropy (Hfeature) and compute the following:

$$Hfeature(x) = -\sum_{i=1}^{m} Pilog2(Pi)$$

For each feature ∈ {Source IP, Source Port, Destination IP, Destination Port, Protocol, Initial Flags, Union Flags, Reverse Initial Flags, Reverse Union Flags, End reason} in addition to features that combine {Destination IP and Port, Destination IP and Initial Flags, Source and Destination IP, Source IP and Initial Flags}. These 14 entropies are the input to their machine learning model. Their machine model classifier is a fully connected feed-forward neural network, consisting of an input layer, three hidden layers and an output layer. The hidden layers have 100, 30 and 100 nodes respectively where all of them uses the ReLU as the activate function. The output layer uses Softmax as the activation function and consists of the following five categories: no attack, DDoS attack, port-scan attack, Peer To Peer (P2P) Denial of Service (DoS) attack and network scan attack. They've run some experiments on the MIT SuperCloud cluster, on nodes with Intel Xeon E5 processors and 64Gb of RAM per node. Of the MAWI dataset, they used 36 days for training the model and 17 for validation. Their classification set was generated using the Intrusion Detection Dataset Toolkit (ID2T) to directly inject malicious network traffic on samples. This tool makes it possible to adjust the frequency and which type of attacks to ingest. It took about 3 minutes to train the model, performing a total of 2000 epochs, corresponding to an accuracy of 97% and 95% on the training and validation dataset. They've achieved impressive results: their model has about 90% accuracy to detect DDoS and port scan attacks with just 4% of the total network traffic being malicious. They state that their model can be applied in real-time network.

Often the network traffic is encrypted or involves an unknown protocol, making it a challenge to analyze those network packets. So, instead of performing a Deep Packet Inspection (DPI) on the encrypted payload or of unknown protocols, a Deep Flow Inspection (DFI) is performed, making it achievable to analyse the traffic. The Y. Guo et al. [23]

proposal, consists of a framework that analyses the network traffic with the conjunction of both DPI and DFI approaches to detect DoS, probe and privilege escalation network attacks. Their framework consists of 3 phases. The first phase performs a deep packet-level (DPI) inspection by resolving the protocol headers, fingerprints of the application layer, etc. In the second phase, they adopt data-mining for the DFI. In the last phase, the final result will be the detection result of the DFI in case of encrypted traffic or unknown protocols; otherwise, a comparison is made for DPI and DFI and when the fingerprint of both matches, a result can be acquired. They've chosen the C4.5 data-mining decision tree as the classifier for the DFI. As for the data, they used the KDD Cup '99 dataset where the '10% KDD' was used for the training of the model and the 'KDD Corrected' for the testing. They've chosen the following features for their data-mining model: duration, protocol-type, service, src_bytes, dst_bytes, num_failed_logins, loggen_in, root_sheel, num_access_files, num_outbound_cmds, count, Serror_rate, srv_rerror_rate, same_srv_rate, dest_host_srv_count, dest_host_same_src_port_rate and dest_host_rerror_rate. According to their results, their model isn't very valuable. For the detection of probe and DoS attacks, the model has an accuracy of 72% and 92%, while for privilege escalation the accuracy is only 4%.

A. Ulmet et al. [24] note that Wireshark displays data as a table, making it difficult and time-consuming to get an overview and focus on the significant parts. Also, each time Wireshark is initialized, the process of loading the files is repeated, taking a large amount of time to open big files. Therefore their goal is to develop a web-based alternative to Wireshark. That way, as the service is easily accessible more users will try to analyse the data. As the PCAP file contains a lot of information and users may not need all the data, the authors implement techniques to reduce the amount of data to upload and store on the server. At first, the user must select the PCAP file to be uploaded; when the file is chosen the system collects statistics of the entire PCAP file, so the user may select which protocols to filter and which IP addresses are internal (to ease on the analysis). In the uploading phase, the network packets are uploaded to the server as small bits (data chunking) with their payload omitted. Each time a chunk of data is uploaded and stored in the server, the server will return the data needed for visualization back to the client, giving immediate feedback to the user so it may start to analyse the data while the rest of the PCAP file is being uploaded. The other method of uploading would be process chunking, which is especially useful for providing an early prediction of the result. As for the technologies they use a MySQL database to store the data; the visual interface is processed on a Spring Boot Java Server that is connected to the database; the communication between the server and client is handled by WebSockets using SockJS and Stomp to lower the latency; the front-end is supported by React.JS, Material-UI and D3.JS. The authors tested their system with two different PCAP files. The first dataset has nearly 29,000 packets with 19 MB in size; the second has 240 packets with 70KB in size. The goal of their experiment is to determine a suspicious IP address and then use the same filters in Wireshark to inspect the raw data. In the first case, the authors state that the system is capable to perform the tasks of finding suspicious patterns, time frames and IP addresses. In the second case, they conclude that with the filtering mechanism of their system, the analysis time can be reduced by only looking at the significant parts of the data.

X. Ye et al. [25] analyse the network activity by detecting anomalous network behaviours based on a host's social relationship (to whom it communicates) interaction patterns. Their work captures network traffic as a traffic activity graph and analyses group activities corresponding to the community evolution, considering both structural and temporal properties of network behaviours. This is unlike most other studies, which ignore the temporal changes and only focus on static graphs. Their system performs the analysis only using the source and destination IP addresses, reducing the data volume and computation complexity. They discovered that analysing the network evolution at a different point in time is useful to monitor the network, as hosts often act as a group on network attacks (e.g. DDoS, botnet, etc.). Using the network evolution concept, they can compare and determine the changes in the number of each evolution event between two snapshots. With this approach, they can capture the structural properties of group activities and calculate their absolute and relative changes. Before a network attack, often there is a probing behaviour on the network to identify hosts and services, but this kind of attack does not impact the network operation, thus not attracting much attention from the security administrators. Being this the main reason for their study, they pay particular attention to detecting and defending group-oriented attack patterns by analysing the social relationship between hosts. Their system consists of three phases. The first phase, data preparation, cleans the data, extracting both the source and destination IP address, processes it, stores it and then applies Apache Spark GraphX to construct a graph model. The second phase, mining TAGs, obtains the host community in the given graph, with a fast unfolding algorithm to discover every group. After building the profiles of the group activities from consecutive time steps, a standard dynamic evolution can be defined. In the last phase (anomaly detection), the group evolution events that deviate from the normal pattern are considered anomalous. The authors made some experiments using the CTU-13 dataset as the data source and their system showed an average accuracy of 99.91% and 97.84% of precision. They then concluded that their system can effectively identify group activities and accurately detect anomalous hosts.

Domain Name System (DNS) protocol is being used to support stealth botnet communications between the bot and its Command and Control (C&C) center. This communication starts with the bot sending a DNS query to the C&C center directly, without using the organization name resolver, and with the payload of the query being hashed to encode the content of the communication. H. Ichise et al. [26] propose a framework with the aim of the detection and blockage of anomalous DNS traffic by analysing archived Name Server (NS) records history. To acquire a list of allowed DNS servers, using TCPDump they captured traffic and analysed it using DPKT (a Python framework) to construct their white-list dataset and stored it on a MariaDB database. They use Software Defined Network (SDN) technology, namely the OpenFlow switch protocol and a controller on their system, for the detection and blockage of unauthorized name resolvers. When a client sends a DNS query, the OpenFlow Switch sends the query packet to the controller; the controller will then check in the database if that IP address is allowed, returning the response to the switch. Depending on the response from the controller to the switch, the switch will then redirect or drop the packet. The authors performed a functionality experiment, without worring about the performance of the system. As the result of the experiment, their system blocked a DNS query done to 8.8.8.8 (Google DNS server) but not to 8.8.4.4 (Google DNS alternative server), proving their system isn't a viable option as it may block important legitimate traffic. Nevertheless, the problem with this system relies on the construction of the white-list: performing packet capture on the network to obtain the list isn't a good approach as the network may already have been infected and communications between the bot and its C&C center may be already happening, adding the malicious name resolver to the white-list. A good fix could be to redirect the DNS query to a IDS to further analysis.

2.3 Other application scenarios

In addition to security auditing, network traffic collection may also be used for many other purposes. Managing the network is one of them: by analysing the retransmission rate of the packets, loss of connectivity or network failures, it may be determined the QoS of the network to figure out if it's necessary to add more routing points throughout the network.

Network data may also be used to compute an estimation of the occupancy of a room. E. Longo et al. [27] designed a system with cheap Wi-Fi sniffers to estimate the occupancy of several rooms by analysing the Wi-Fi Probe requests and probing Bluetooth scan frames.

Compliance enforcement is another potential application field using network packets, making it conceivable to set network policies [28], investigate compliance violations on an enterprise network or even investigate the compliance of devices with their Privacy Policy Agreement (PPA), like in the case of the A. Subahi et al. study [29], that sniff the data packets moved between Internet of Things (IoT) devices and the cloud and check if the devices comply with their PPA.

Finally, user behaviour may also be analysed using network data. P. Boonyopakorn [30] proposes a monitoring system to analyse the user behaviour on the network by analysing the network data.

2.4 Functional and security requirements

Accordingly to H. Lin et al. [31] there are a set of functional and security requirements that an IDS should meet to be recognized as a solid system able to operate in modern networks. The functional requirements are the following:

- It must be able to collect required security-related data;
- It must be able to know when to collect the data;
- It must be capable of dynamically knowing which kind of data to capture (what filters to use);
- It must be capable to export the data to other systems;
- It must be able to manage and control the data;
- It must be efficient and stable when collecting the data, a very important requirement because missing packets can compromise the analysis of the network;
- It must be flexible and scalable;
- It must not use too many resources to not affect other local operations;
- It must be automatic in terms of adaptability, in case of modifications on the network structure;
- It can not destroy the original network system;
- It must be universal and generic, supporting multiple applications scenarios;
- It must not produce new data that may affect the accuracy of the collected data;

- It must be able to store collected data in a storage medium.

Concerning the last functional requirement, it should be pointed out that a system may still be solid even if it doesn't store the collected network data. On a near real-time analysis system, the captured data may only be needed for a brief couple of seconds, until the system has time to process that packet. After the packets are processed (feature extracted or subject to other kinds of operations) it may not be necessary to have them stored anymore. By taking this strategy, huge amounts of resources will be saved but it won't be possible to perform the analysis on network data from previous days.

The system must also make sure that the data that was collected was not changed during the transmission or storage, maintaining the integrity of the data to prevent the analysis of modified data that would invalidate the whole system operation. H. Lin et al. [31] also presents a set of security requirements that a data collection system should meet to minimize the probability of working with adulterated data. The security requirements are the following:

- It must be able to prevent data loss and ensure data integrity during the capture and transmission;
- It must protect the user privacy;
- It must ensure the security of collected data and be able to prevent any data leak;
- It must be able to verify the integrity and authenticity of the collected data;
- It must protect the data against unauthorized users.

2.5 Packet capture

Cyberattacks perpetrators usually make efforts to cover their tracks during an attack. Security researchers can find new ways to prevent cyberattacks the same way attackers can adopt anti-forensic techniques trying to remain undetected and without leaving traces

[32]. Log files can be used to detect some attacks, such as massive unauthorized accesses or failed logins. However, they are not enough in most situations, since it is not possible to detect all kinds of attacks and there is also the possibility of those getting modified or erased, eluding the security team scrutiny. There is only one thing that attackers (or anyone else) can never change or purge and that is the network traffic. As it can never be changed or removed, it's the best candidate to perform a full-depth analysis of the network, trying to identify who the attackers are, when the attack took place, for how long, with which tools, and what was transferred. Nevertheless, as the network traffic is volatile information, meaning it only exists while being transmitted, it's necessary to capture it and store it in real-time [33]. As expected, the amount of network traffic is considerable, easily reaching terabytes worth of space in a matter of seconds (depending on the infrastructure), making its collection and storage a very expensive operation [16]. Depending on what the security needs to comprehend about the network, it must select which type of data must be stored, from a byte in a header to the full-packet capture.

2.5.1 Network data tiers

Network flow data tier represents streams of network packets by generating one flow record for all packets seen on an observation point over a period of time [34], [35]. A single flow is constituted by a set of packets with the same source and destination IP address. It is the tier that requires the least amount of storage space by dropping payloads and most of the header information. However, it's also the one that less information has, limiting the analysis of the network data. An IP network flow record must define the following properties [36]:

- one or more packet header field (IP address), transport header field (port number) or application header field;
- one or more characteristics of the packet itself;
- one or more fields derived from packet processing.

To cope with the previous limitations, it is possible to keep additional information. The augmented network flow data tier adds information that may be extracted from the header or payload or derived from the flow/packet characteristics (e.g. passive operation system fingerprint).

An alternative approach is the full packet data tier that captures the packets that travel from one endpoint to another. This means that all the payload, as well as headers, will be captured, which requires much more space than the other data tiers. It also requires more computational power for the analysis.

Table 2.1 helps to understand what is possible to discover, during an investigation of an attack, for each data tier [34].

Table 2.1: Network data usage [34]

	Properties						
Data tier	Who	How Much	When	How Long	Using What	Transferring What	How
Network Flow	✓	✓	✓	✓			
Augmented Flow	✓	✓	✓	✓	✓	✓	
Full Packet	✓	✓	✓	✓	✓	✓	✓

2.5.2 Data capture tools

There is a vast number of different tools to perform network packet capture, some of them are software-based and others are hardware-based.

Software-based packet capture tools consist of several subsystems. The packets flow starts at the Network Card Interface (NIC) (hardware subsystem) moving to the Kernel space subsystem (device driver and operating system) to reach the userspace subsystem, containing the packet capture library and application. If any problem occurs in any of those subsystems, packet loss will most likely occur [31].

Hardware-based tools are usually more expensive than software-based ones, as they are physical devices. Data Acquisition and Generation (DAG) cards are effective devices to capture network packets in high-speed networks. Those cards can even apply filters at the hardware level, further improving the performance [37]. In routing devices that have the packet-forward functionality, when configured, they send a copy of every packet that crosses that routing device to a specific physical port. The advantage of using such an approach is that enabling port mirroring does not require the modification of the current network infrastructure; it's only necessary to plug a capture device in an available physical port and configure the port mirroring to that port. Inline taps provide a full view of the network packets that move through the wire without any impact on the network data. However, setting up this tool requires rupturing the connection. The advantage in using such a tool compared to port mirroring is that port mirroring may have some packets dropped if the routing device's buffer is full or if the packets are malformed, while in the case of inline tap, that problem does not occur.

2.6 PCAP file format

The majority of packet capture solutions, including TCPDump, stores the raw data into a PCAP file format. Each PCAP file lies a fixed-length size global header containing information about the file and the format of the packet records. The multiple fields that the PCAP global header has are presented in detail on table 2.2.

Table 2.2: PCAP global header format, adapted from [38]

Designation	**Size (bits)**	**Description**
Magic number	32	Unsigned value used to identify the file endianness and also if the timestamps on the file are in seconds and microseconds or seconds and nanoseconds.
Major version	16	Unsigned value that represents the major version of the PCAP format.
Minor version	16	Unsigned value that represents the minor version of the PCAP format.
Reserved1	32	Currently unused bits, should be filled with zeros.
Reserved2	32	Currently unused bits, should be filled with zeros.
Snaplen	32	Unsigned value that represents the maximum size of the packet; normally it's a standard value (65535).
LinkType	32	Unsigned value that identifies the link layer header type; the different types can be seen at `https://www.tcpdump.org/linktypes.html`.

The PCAP global header is followed by zero, one or many records (figure 2.1), where each record is composed of a fixed-length size packet header and the actual network packet [39]. The packet header is essential to allow the retrieval of the correct information: it's possible to only save a specific size of the packet instead of the full content, therefore, to retrieve that packet from the PCAP file, how does the reader application know the actual length of the packet? If all the packets were not truncated it would be possible by dissecting the packet and getting the length of each layer to get the entire packet; but as they may be truncated it's required extra information to know the exact length. And that's the use of the packet header: it not only tells the actual length of the packet but also other information like the timestamp of that packet capture. A detailed description of the fields of the packet header can be visualized on table 2.3.

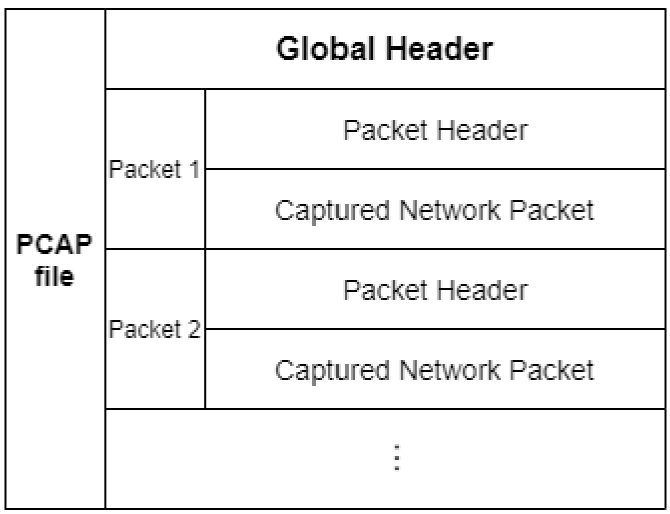

Figure 2.1: PCAP file format, adapted from [39]

2.7 Tools

Sometimes, selecting the right tools for a certain task is not easy. Some tools perform certain task better than others; some are more interoperable than others; some are the best but for a specific context they won't work; etc. When implementing a system to perform the analysis of huge chunks of data, selecting the appropriate technologies and tools is halfway towards a solid system [40]. Therefore it was conducted a literature review

Table 2.3: PCAP packet header format

Designation	**Size (bits)**	**Description**
Timestamp (seconds)	32	Unsigned value integer representing the time when the packet was captured. The value is the number of seconds that have elapsed since the epoch time (1970-01-01 00:00:00 UTC).
Timestamp (microseconds or nanoseconds)	32	Unsigned value integer representing the number of microseconds or nanoseconds elapsed since the seconds specified before. The specification of the value is microseconds or nanoseconds is defined in the global header of the PCAP file.
Packet captured length	32	Unsigned value that indicates the number of octets captured from the packet.
Original packet length	32	Unsigned value that indicates the length of the packet when it was transmitted through the network.

on the technologies and tools to help select the most adequate ones for the job. Provided they were open-source. This way, the risk of the necessity of changing a tool while the system is being implemented is reduced.

2.7.1 Capturing tools

The selection of the best network packet capturing tool is probably the hardest one. Many tools are available to the public as open-source, but the majority can't reach multi-Gbps rates.

Scapy is a framework in Python very easy to use but, as Python is a high-level interpreted programming language, it does not produce very performant applications. Then, Scapy can't reach multi-Gbps capture rates and sufferers from very high CPU usage, leaving no available resources to perform other operations like writing to disk or send the information to another system [41].

nProbe is a great capturing tool that uses PF_Ring to reach 100 Gbps rates while capturing packets [15] but it's not free and thus can't be considered for this work.

D. Álvarez et al. [42] performed a CPU usage comparison between TCPDump, Wireshark and Tshark when sniffing the network. While TCPDump keeps an average of 1% of

CPU usage, Wireshark and Tshark use 100% and 55% respectively. As such, TCPDump was the selected tool to perform the network packet capture. Once the tool works on top of the libpcap framework, it uses a Zero-Copy mechanism, reducing the data copies and system calls, hence improving the overall performance.

2.7.2 Transportation tools

Apache Kafka is composed of servers and clients that perform event streaming between each other. Event streaming is the practice of capturing data in real-time from one or multiple sources and storing it for later retrieval. It works based on the Publish-Subscribe model, where producers publish to the distributed queue and consumers subscribe to get the data when they and the data are available [43].

Kafka is run as a cluster of one or more servers that can be placed on multiple datacenters. Some of these servers, designated as brokers, form the storage layer, while others continuously import and export data as event streams to integrate Kafka with other existing systems. A Kafka cluster offers fault tolerance in the case any of the servers fails. If that befall, other servers will take over their work, ensuring continuous operation without data loss [44].

The clients allow writing distributed services that read (consumers), write (producers) and process streams of events in parallel, offering the same perks as the servers (fault-tolerant and scalability).

Apache Kafka has the following functionalities [45]:

- Events are organized and stored in topics and can be consumed as often as needed;
- A topic can have zero or multiple producers and consumers;
- Topics are partitioned, allowing the disperse of a topic over several "buckets" located on different (or the same) Kafka brokers;
- A topic can be replicated into other brokers, this way, multiple brokers have a copy of the data, allowing the automatic failover to these replicas when a server fails;

- Kafka performance is constant regardless the data size.

Apache Flume offers the same Kafka perks but it uses the “push” model, where instead of being the consumer to fetch the data, it’s the service that forwards the data to the consumer [46].

D. Surekha et al. [47] and S. Mousavi et al. [15] uses Apache Kafka on their systems because they defend that it is a fast, scalable and reliable messaging system with good throughput, replication and fault tolerance.

Apache Kafka is selected for this work not only because many people use it but also because Apache Flume may flood the messages as it pushes to the consumers regardless if they are ready or not, instead of being them to fetch the data as it happens on Kafka.

For a public cloud solution, Amazon Kinesis is a good alternative as it can handle hundreds of terabytes per hour of real-time data flow [48].

2.7.3 Storage tools

HDFS is one of the 60 components of the Apache Hadoop ecosystem, with the ability to store large files in a distributed way, dividing the information in chunks across multiple nodes, offering reliability and extreme fault-tolerance. It is based on the Google File System [49] with the design of write-once-read-many [50]. This system is composed of two main entities, one or more NameNodes and DataNodes. The NameNode stores the metadata of the files and where the files’ chunks are located. It is also responsible to inform the clients in which DataNodes the necessary chunks are stored. Files chunks get replicated across the DataNodes reducing the risk of system failure in case of a DataNode failure. The DataNodes are responsible for the storage and retrieval of data blocks as needed [51]. Some examples where HDFS is used follow.

K. Madhu et al. [52], S. Mishra et al. [53], K. Aziz et al. [54] and R. Kamal et al. [55] all perform real-time data analysis on tweets from Twitter that are stored in HDFS. S. Kumar et al. [46] uses HDFS to store in real-time massive amounts of data produced by autonomous vehicles sensors. J. Tsai et al. [56] analyse in real-time road traffic to estimate future road traffic while the data is in HDFS.

Ceph is a reliable, scalable, fault-tolerant and distributed storage system [57]. It allows to not only to store files but also objects and blocks.

Gluster File System is a scalable file system capable of storing petabytes of data in a distributed way [58].

C. Yang et al. [59] made a study comparing the HDFS, Gluster FS and Ceph performance while writing and reading files. According to their results, the authors find that the performance of HDFS is better than the other two.

M. Tanaka et al. [60] project is to improve the performance of telescope data processing, focusing on the scalability of parallel I/O usage. They discussed the following tools to store the information: HDFS, IBM Spectrum Scale (a high-performance scale-out parallel file system), Gluster FS and Gfarm FS (a distributed file system for large-scale cluster computing). IBM Spectrum Scale provides high I/O performance by stripping across nodes, while the remaining tools scale out by using the storage of worker nodes. In their study, they compared IBM Spectrum Scale and Gfarm FS and they concluded that Gfarm FS scales better when they have more than 16 nodes. They also point that HDFS wouldn't fit their needs as it can't perform random writes, with Gfarm FS being a good alternative in case of the need for random writes on the files.

S. Paul et al. [61] performed a study to compare the read and write operations on different HDFS data blocks sizes (64MB, 128MB and 256MB). According to their study, the best result is 256MB of block size. They also conclude that if they extend to higher block and file size, the read and write operations will further improve.

HDFS was selected to support the distribution of files not only because it is vastly used but also because it provides the necessary perks to provide a solid scalable distributed file sharing solution. Also with version 3 of Hadoop, multiple NameNodes are supported reducing the risk of system failure. There are also other storage tools like Apache Cassandra, a distributed NoSQL database and Apache HBase, a big data distributed database that supports tables with billions of rows and millions of columns, both of these tools work on top of HDFS [62].

2.7.4 Stream process tools

Apache Spark is subdivided into two main modules: the Apache Spark Streaming and Apache Spark engine. Apache Spark Streaming provides a high-level abstraction (DStream) representing a continuous flow of data [63]. It receives the data from sources such as Apache Kafka, Amazon Kinesis, etc. and sends the data to the Spark engine as micro-batches to further processing [40]. Apache Spark is implemented in Scala and runs on the Java Virtual Machine (JVM). It provides two options to run algorithms: i) as an interpreter of Scala, Python or R languages that allows users to run queries on large databases; ii) is to write applications on Scala and upload them to the master node for execution [64]. Some examples where Apache Spark is used follow.

S. Mishra et al. [65] proposed a framework to predict congestions on multivariate IoT data streams on a smart city scenario using Apache Spark to receive and process the data from Apache Kafka. A. Saraswathi et al. [66] did also use Apache Kafka and Spark to predict road traffic in real-time. Y. Drohobytskiy et al. [67] developed a real-time multi-party data exchange using Apache Spark to obtain the data from Apache Kafka, process it and store it into HDFS.

Apache Storm is a free, open-source real-time computation system capable of real-time data processing [68] just like Apache Spark Streaming. J. Karimov et al. [69] and Z. Karakaya et al. [70] both perform an experiment comparing Apache Storm, Apache Flink and Apache Spark. With their results, they've concluded that Apache Spark outperforms Apache Storm, being better to process incoming streaming data in real-time. Between Apache Spark Streaming and Apache Flink, the selection is more difficult: they both have their pros and cons and similar benchmark results.

According to the experiments of M. Tun et al. [19], the integration of Apache Kafka and Apache Spark Streaming can have a better processing time and fault-tolerance on huge amounts of data.

Apache Spark Streaming is the tool selected as it can have a good integration with Apache Kafka, supporting real-time operations. Also, it uses in-memory computation to perform stream processing and it recovers from node failure without any loss, something that Apache Flink and Apache Storm aren't able to offer [19]. Besides, data can be acquired from multiple different sources like Apache Kafka, Apache Flume, Amazon Kinesis, etc.

2.7.5 Data process tools

Apache Hadoop MapReduce (based on Googles' MapReduce [71]) is a framework for writing programs that process multi-terabyte datasets in parallel on multi nodes offering reliability, as well as fault-tolerance [72].

T. Sirisakdiwan et al. [73] introduces an Apache Spark framework for multiple heterogeneous data streams. They also perform experiments to observe which Spark job scheduling is better for real-time processing, concluding that FAIR is faster than First In First Out (FIFO).

D. Jayanthi et al. [74] compare the computation between MapReduce and Apache Spark. With their experiment, they reported that Apache Spark overcomes the processing speed drawback of MapReduce.

Apache Spark is up to 100 times faster than MapReduce since it uses in-memory processing for large parallel processing [19] while MapReduce performs disk-based operations. MapReduce' approach to tracking tasks is based on heartbeats causing an unnecessary delay while Apache Spark is event-driven [75].

Apache Spark is the tool selected as it focuses on the processing speed, while MapReduce on the massive amounts of data [63]. Besides, Apache Spark contains a vast amount of libraries to support data analysis.

There are several scientific and technological approaches that can be used to overcome the challenges that real-time capture and analysis pose. The next chapter highlights the approach followed in this work.

Chapter 3

Approach

This chapter will manifest the approach adopted to solve the original problem, presenting the system architecture and explaining how it will operate.

3.1 Proposed system

It is important to design a system that respects the functional and security requirements defined on section 2.4. This system (figure 3.1) was designed with the idea of fully horizontal scalability with an easy way to add more physical resources whenever necessary. If more packet capture devices are necessary it should be desirable to just plug in new ones. Conversely, if more analysis services or algorithms are necessary, it should be simple to just add them. That's why the proposed architecture is also designed with the modular principle in mind, subdividing the system into sub-systems allowing the easy adjustment of a specific sub-system without changing the functionality of the rest of the system. Furthermore, it also makes feasible the addition of more modules.

3.1.1 Network traffic capture module

This module captures the network traffic, as it flows through the routing devices. L. Sikos [76] describes the four main ways to capture network traffic from switched networks:

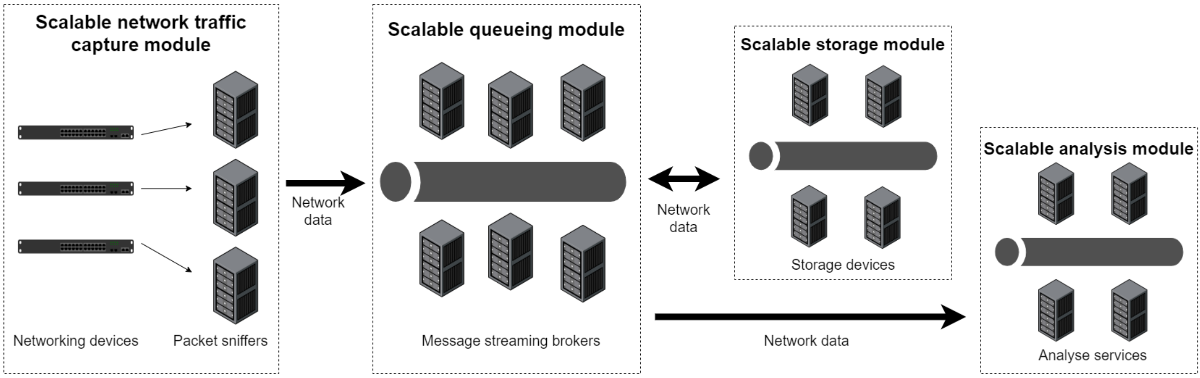

Figure 3.1: Proposed system - components diagram

port-mirroring, hubbing out, inline tap and Address Resolution Protocol (ARP) cache poisoning. But the last 3 goes against the functional requirements since hubbing out would need to change the original network infrastructure and ARP cache poisoning would affect the network behaviour. The remaining alternative is to use port mirroring. However, not every switch or router supports this functionality, which has to be considered in a case-to-case scenario.

Therefore, the port mirroring approach is used to send the network traffic to the capture device (also known as network probe). The number of capture devices depends on the situation. Nevertheless, the strategy is to place a capture device on switch devices that possess endpoints connected to it. This way, in the case of Network Address Translation (NAT), the system does not lose any information and know exactly from which host the packets came from. In this module, only raw data is extracted, without any processing on it.

3.1.2 Queuing module

While the network packets are being captured, they need to be transported to a central storage or processing service. Some systems capture and store the packets locally on the capture machine and only after they get transferred to another location for further analyse (usually in PCAP format). However, this approach is not ideal, due to the delay between the capture and the analysis. In fact, it does not allow near real-time assessment, delaying the discovery of an attack that already happened, nulling the opportunity to prevent it. An approach to achieve near real-time analysis and also prevent an attack or minimize its damage is to capture the packets and send them right away to another service. This way, while a service is capturing packets and sending, another is processing and analysing the data. Considering the increasing size of organizations' IT infrastructure, this is one solution towards near real-time analysis. But, to achieve this goal the system must have a way to transfer the data between services in a parallel manner.

This module objective is to provide mechanisms to have multiple producers (who write the data) and consumers (who read the data) synchronized. This way, services send data to the distributed queuing system while others consume it. Nevertheless, data transportation must not flood the network. Moreover, if the queue is at full capacity and can not keep with the data rate from the producers, the producers must have a way to store the data locally and only send it when the queueing system is available.

3.1.3 Storage module

With a distributed queuing system, the data consuming hosts (analysis modules, classification, estimation, prevision, etc) consume the data directly from the distributed queue in near real-time. The queue system will persist the data for a predefined period of time (may also be set to unlimited time) and deletes the oldest records when it's close to running out of space. Consequently, it can't be used to persistently store the data, thus the need for a data storage module. This module will consume the data on the queuing module and store it with the appropriated meta-data. This way if a service requires data that is no longer available in the queuing system, the storage system will upload the requested data to the queue so other services may acquire it.

3.1.4 Analysis module

After the system possesses a continuous set of information available, the next step is to analyse that data. The approach in this module is to consume the data in the queuing module and process it using any kind of service to aid in the analysis (services such as machine learning classifiers, dashboards, etc.). As the data is in a queuing module, it's possible to split the data throughout different host machines that are running the same application, therefore reducing the computational load on them.

3.1.5 Technologies

TCPDump is used to capture all network packets in promiscuous mode. At the same time that is capturing network data, it's publishing it on the Apache Kafka cluster queuing system. While the data is on the Apache Kafka sub-system, Hadoop HDFS will read from it and store it persistently to not lose any relevant information. While the data is being saved into HDFS, Apache Spark is also consuming the data in parallel, to perform the analysis in near real-time. Figure 3.2 gives an overview of how the technologies are distributed across the system.

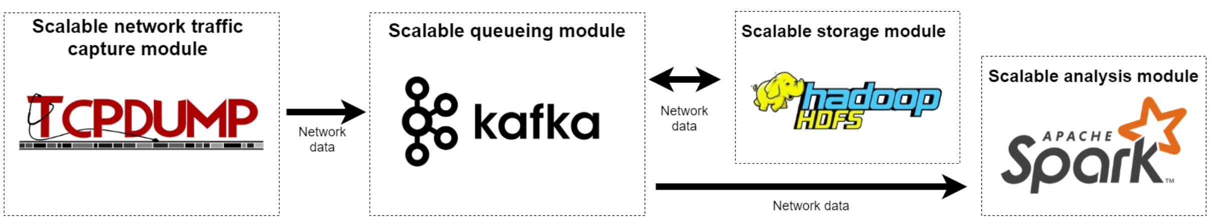

Figure 3.2: Proposed system - technologies

3.2 Domain Generation Algorithms

Domain Generation Algorithms (DGA) are used by perpetrators to generate a large set of domains (also known as malicious domains) so they may control their infected hosts [77]. Even if a domain name gets blocked or is taken down, the infected hosts will just use another. There are multiple lists of detected malicious DGA domains that can be consulted to check if a given domain is malicious or not (also known as benign domains); but, consulting those lists in real-time would be time-consuming; additionally, those lists are uniquely useful to detect domains that were previously detected; in case a non-black-listed malicious domain tries to communicate with a malicious host, the analyzer won't be able to detect it. Therefore the system must be able to detect malicious domains that were never detected previously. To do so it must use an approach that is able to self-learn, like one based on machine learning algorithms. Some works [78], [79] perform the detection of such domains using machine learning with features. The problem with this approach is that extracting such features is time-consuming and even worst the perpetrators can create a DGA that takes into account these features in order to lower or null the accuracy of detection models. The approach in this system is to build a featureless machine learning model where the only input required is the domain name; this way the model can operate in near real-time and it solves the issue of the perpetrators lowering the detection accuracy of the machine learning model. As the model will operate in near real-time, when the analyzer detects a DGA malicious domain it can send notifications to the system administrators or even block the connection (although blocking the connection, is not a truly good approach as that information could be crucial in finding other infected hosts). It should be noted that this DGA malicious domain detection is only a module of the analysis system, which supports multiple different modules executed in parallel.

With the main architecture defined, the implementation also requires to consider the limitations of the infrastructure and the configuration of all the modules within. This will be the subject for the next chapter.

Chapter 4

Implementation

This chapter will explain the implementation of each system module, describing the problems encountered and how they got solved. It also presents the characteristics of the test-bed used to host the services of the system.

To respect the functional and security requirements described in section 2.4 it was created a fictional Certificate authority (CA) to generate certificates for the entire system; this way, all the hosts could communicate by Secure Sockets Layer (SSL) security protocol by trusting the CA.

4.1 Scenario

The architecture was deployed in the laboratory of infrastructures and communications at ESTIG. There are two different networks present in the laboratory: one network (A) is where the capture of the packets is performed and the other (B) is where the system services are connected to; htis is similar to a real-scenario situation.

Network A has 2 physical devices connected to a Cisco Catalyst 2960-S switch, where both of these devices have a gigabit NIC. The switch is configured with a Dynamic Host Configuration Protocol (DHCP) service and the port mirroring is active, sending the traffic of all the ports to an output port; the configuration of the Switch can be found on appendix C listing C.1.

Network B is constituted by a total of 14 physical computers where 13 of them have the same system specifications (listed on table 4.1) and the remaining one is the network probe with different system specifications (table 4.3). In each of those 13 devices is running a virtual machine (configurations on table 4.2) that is hosting a service node. All of these devices are connected to a Cisco Catalyst 2960-L Switch.

In figure 4.1 it's shown a logical representation of the test-bed components in the laboratory. The capture module is composed of 1 machine, the queuing module by 5, the storage module by 3 and the analysis module by 4. The Apache Zookeeper, Apache Spark Master and Apache Hadoop Namenode are all running on the same virtual machine since they do not need much computational power as these services only coordinate the cluster. In a real scenario, those services should operate on separate machines and with more than one instance, so they offer some fault tolerance. The network capture device, aside from being connected to Network B to send the data, it's also connected to the switch of Network A, on the port-mirroring port so it may receive the network packets of network A.

Table 4.1: Laboratory computers system specifications

Processor	Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, 3192 Mhz, 6 Cores, 12 Logical Processors
RAM	16 GB
NIC	Intel(R) Ethernet Connection (2) I219-V, 1 GB rate per port
Storage device	SSD with 460 GB in size
Virtualization software	VMware Workstation Pro 16.1.2

Table 4.2: Laboratory virtual machine specifications

Processor	2 processores with 2 cores each
RAM	8 GB
Network adapter	Bridged mode
Storage device	50 GB (preallocated)
Operating system	Ubuntu Server 20.04.3 LTS

Table 4.3: Network probe system specifications

Processor	Intel(R) Core(TM) i7 CPU @ 2.67GHz, 4 Cores, 8 Logical Processors
RAM	24 GB
NIC	Realtek 8111C PCI Express Gigabit Ethernet Controller (2)
Storage device	SSD with 240 GB in size
Operating system	Ubuntu Server 20.04.3 LTS

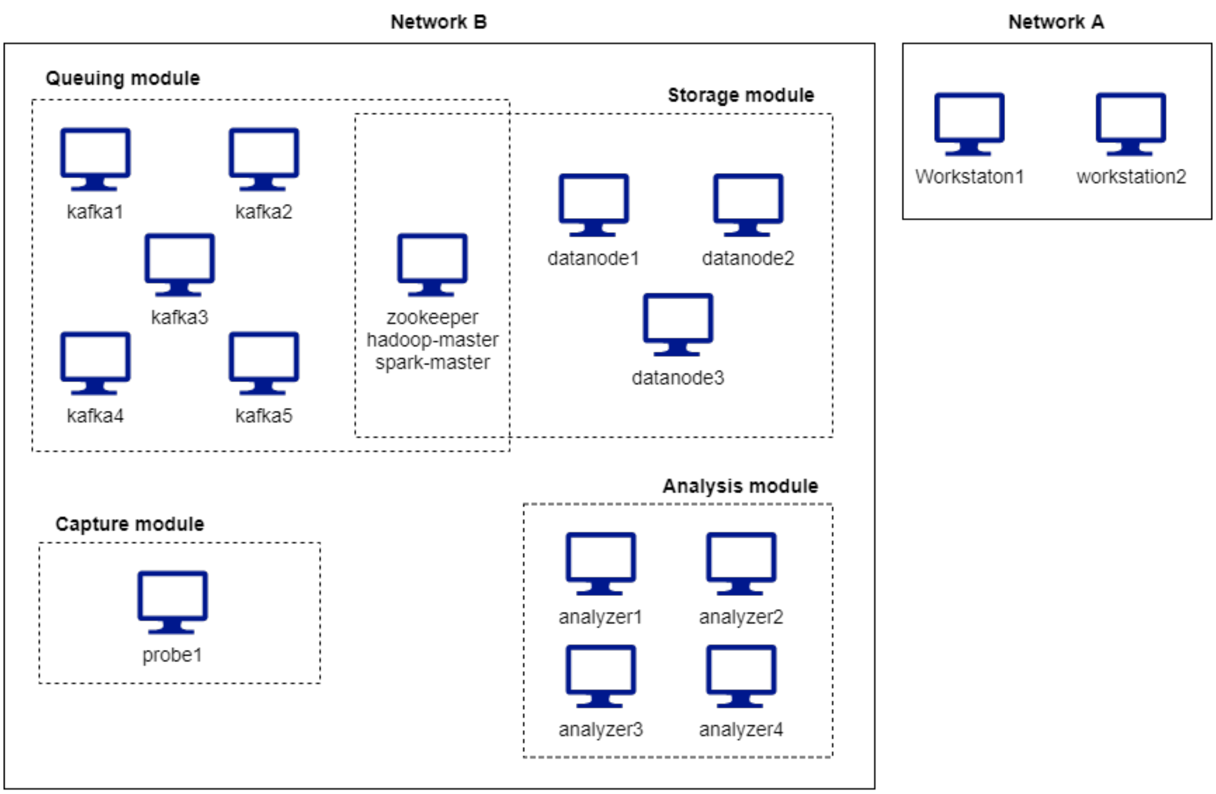

Figure 4.1: Computer laboratory diagram

4.2 Packet capture module

The main requirement for this module is to perform full packet capture at 1 Gb/s without losing any network packets both on the header-only capture and full-packet capture. Throughout the development of this module, multiple application versions were developed, improving the performance and results from each version. A comparison between the 3 application versions can be found in section 5.1.

The first version was developed using Python version 3.9, the procedure was to open TCPDump through a pipeline, read the standard output and write into the queuing module. Needless to say that the application when up against a stress test it lost almost all the packets both in a header-only and full packet-capture. The problem was, as the application was capturing at a 1Gb/s line rate, it couldn't keep up with the load between getting the data and uploading it to the queuing module. A possible solution was to have a data buffer structure hosted on the machine RAM and two separate threads, the writer and the reader. As the name implies the writer thread is responsible for capturing the network data and write to the data buffer and the reader thread, when available, to read from the buffer and publish in the queuing module.

The second version (see listing E.1 in appendix E) had a little shift in the technology, it was changed from Python to C (standard 17), the code was forked from `https://github.com/jmakov/gulp` and changes were applied to enable the integration to the queuing module. With this improvement, the application was now able to keep up with the load but only in the headers-only capture, in full-packet capture, the reader thread couldn't keep up with the writer even with a data buffer so once the data buffer was filled the application started to lose the packets.

The third version (see listing E.2 in appendix E) appeared to fix the problem from version two, the technology was changed back to Python and now the data buffer was not hosted in the machine RAM but on the disk. The application had the same threads, but this time, the writer would use TCPDump to capture the network packets and write to a file in the local disk, and the reader would read from that file and write to the queuing module. With this change, the application is now able to perform a full-packet capture at 1Gb/s losing a very low amount of network packets (if any).

The problem with version 3 is that it requires the capture machine to possess a large amount of disk space, while version 2 didn't as it hosts the data buffer on the machine's RAM. Now, the selection of the version depends on the requirement of the capture: in the case of header-only capture it is recommended version 2; in the case of full-packet only version 3 will succeed on the task.

The packet capture script accepts a different set of arguments (table 4.4) it; some parameters like Kafka connection configuration and SSL keys are not passed as a parameter but stored in a configuration file that can be easily changed.

Table 4.4: Packet capture application arguments

Argument	**Description**	**Default value**
Interface	The network interface controller where packets will be captured from. It is a mandatory parameter.	
Filter (-f)	TCPDump filter to be used on the capture.	
Snaplen (-s)	Max size of each packet in bytes. Value of 0 means full packet.	0
Topic (-t)	Kafka topic where the data will be writen to.	"packet-capture"
Kafka chunk size (-k)	Size in bytes of each Kafka message before being sent to the cluster.	524288 (512 KiB)

As stated before, the final version of the packet capture script (listing E.2) is divided into two different threads: the writer and the reader.

The writer thread has the following tasks. First of all, it starts TCPDump as a subprocess, with the appropriate application arguments, and opens a pipe to read the standard output of the summoned process. In turn, TCPDump will set the chosen NIC into promiscuous mode and write all the packets that match the filter (if specified) to a file (PCAP format) on the local disk. It's the main thread that commands the write thread when to start and when to stop the capture of the packets.

The task of the reader thread is as follows. First, it opens the file that is being written by the writer thread and will read the file into chunks of a fixed size and publish them to the queuing module until the file doesn't have more data and the capture has stopped. The writer needs to ensure to not publish a chunk where a packet may be split or the analysis module will be unable to analyze the packets (explanation on section 4.3.1). On figure 4.2 it's presented a simple example of how the size of the chunk is calculated for each message to be published. Even if the user chooses that each chunk is 525 bytes, the chunk in this example can not be that size, as it would cut a network packet in half; so for that message, the chunk must be 450 bytes on size and the packet 4 will belong to the next chunk (note that this was only an example, network packets vary in size).

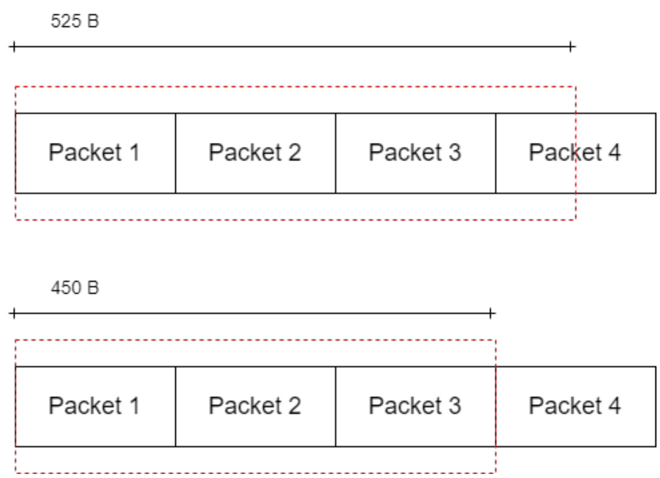

Figure 4.2: Selecting the size of the message to publish

For the reader to perform this operation, it must know how to decode the raw data in the PCAP file, so it knows the size of each packet. The procedure is as follows: the reader will get a fixed-sized chunk (size is specified by the user) from the file and will interpret that chunk. The interpretation consists in reading each packet header of the PCAP file (not to get confused by the network-packet header) and get the length of the saved packet. If the header plus the actual packet fits into the message to be published, then add it; otherwise, stop interpreting that chunk, publish the message, append the leftover bytes to the next message and repeat the procedure.

4.3 Queuing module

For the queuing module, it's used version 2.8.0 of Apache Kafka and version 3.7.0 of Apache Zookeeper. Zookeeper is required to perform the leadership election of the Kafka broker and the topics partition and to track the status of all the nodes in the cluster [80]. In future versions of Apache Kafka, it is planned to remove the need in using Apache Zookeeper and instead use a self-managed quorum.

To aid in the installation of these services, a script has been written to install both Apache Zookeeper and Kafka; this way, when it's needed to insert a new machine to participate in the processing, it's only required to run the script with a privileged user on the new node. The functionalities of the script are the following:

1. Upgrade the system;
2. Download and install the service;
3. Copy the service configuration file;
4. Copy the service manager configuration file; this way it becomes easier to manage the service and it will start automatically when the node boots;
5. Copy the SSL files so the node is accepted in the cluster;
6. Start the service.

Appendix D contains the installation script, the service configuration file and the service manager configuration file of the Zookeeper (listing D.1, D.2 and D.3 respectively) and Kafka (listing D.4, D.5 and D.6 respectively).

4.3.1 Data flow

Messages are only ordered in Kafka per partition and when consuming a topic that possesses multiple partitions nothing guarantees that the message will be retrieved in the same order that it was published. The only way to get the messages from a topic in the same order is if that topic only had one partition, but that eliminates the perks of the scalable queuing message eradicating also the requirements of this system.

On figure 4.3 is represented a simple example of how the producer and consumer work in Apache Kafka. The producer will use a round-robin strategy to publish the messages (publishing in circular order), but that doesn't happen all the time and it may produce two consecutive messages in the same partition; nonetheless, the important concept is that the producer picks the partition where to write the messages. On the consumer, the principle is the same: nothing guarantees that it will also use the round-robin approach and like the producer, it will choose the partition to read from.

As the information is a set of sequential bytes, to be able to read the packet it's necessary to first read the packet header to know how many bytes the packet has and then get that amount of bytes, in case of a packet is split between Kafka messages, its retrieval becomes impossible as the packet header or the packet may be split into two non-sequential messages.

And this is the reason why the packet capture application can not split the network packets between messages, as described before, and can only produce messages with unsliced packets header plus the corresponding packet. Also, as described in section 2.6 the packet header contains the time of when the packet was captured; this way, it's possible to re-order the packets while consuming them, not limiting the analyzers to perform a connection flow analysis.

4.3.2 Message size and partitions

Apache Kafka wasn't designed to handle large-size messages, not being recommended to produce messages above 1 Megabyte (MB). But, what size should the message have to obtain the best performance? To find that value experiments were conducted, comparing the length of 128 Kibibytes (KiB), 256 KiB and 512 KiB; the results and discussion of this experiment can be visualized in section 5.2.

When creating a Kafka topic it's necessary to provide the number of partitions that the

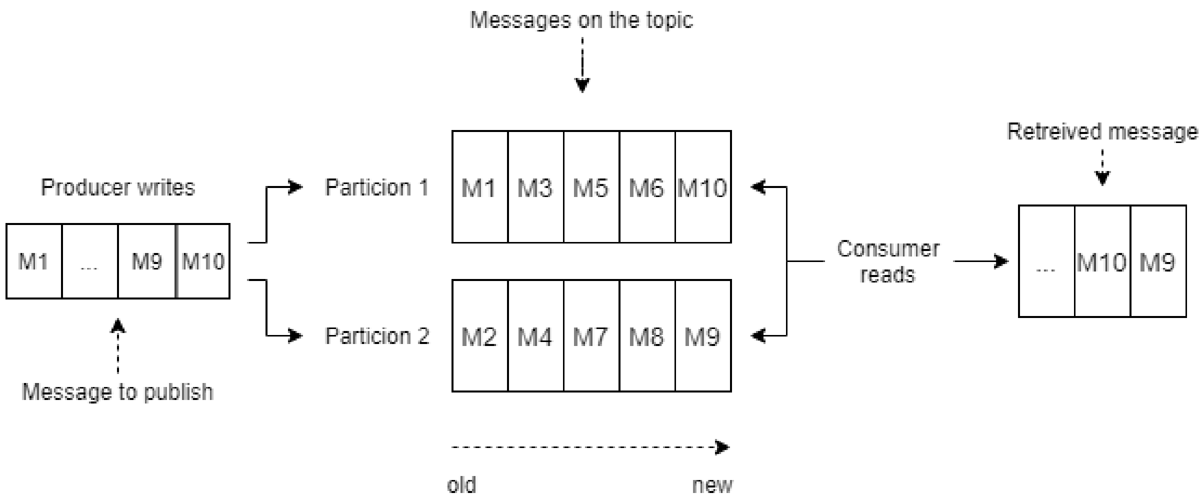

Figure 4.3: Apache Kafka producer and consumer

topic will contain. The partitions may be distributed across Kafka nodes or on the same host; it's Kafka that decides which host will be responsible for the partitions, prioritizing the ones with fewer partitions. But how many partitions a topic should have to give the best performance? In theory, the more the better but is that true in practice and, does it pay off/it's necessary to allocate that extra partition in another Kafka node? Section 5.3 documents experiments to answer these questions, comparing the performance with one to four partitions.

4.4 Persistence storage module

For the persistent storage module, version 3.2.2 of the Apache Hadoop was used. Like Apache Kafka, two scripts have been written to aid in the installation process of the service. One is for the installation of name nodes and the other for the data node. The functionalities of the script are the same as the one for Apache Kafka (section 4.3). In appendix D it can be found the installation script and service manager configuration file of the name node (listing D.7 and D.8 respectively) and data node (listing D.9 and D.10 respectively). Both name and data node share the same configuration files less the security configurations. All of the configurations can be found on appendix D listings, D.11, D.12, D.13, D.14, D.15 and D.16.

A performance experiment was conducted to know the delay of the system regarding the storage of the network data in distributed files. The results and the discussion can be found in section 5.4. The source code of the application that reads from a Kafka topic and stores it into a file on the HDFS cluster can be found on appendix F listing F.1.

4.5 Analysis module

As specified on the system specifications, the analysis module is a cluster that is ready to receive any kind of application to run and retrieve results. There are two different ways to run applications: i) by submitting a Spark job to the Apache Spark cluster, hosted by the analysis machines; ii) to submit a standalone application to one of the analysis machines. For the Apache Spark cluster is used version 3.1.2. To ease in the installation process two scripts have been written, for the installation of the master and worker. Appendix D contains the installation script and service manager configuration file of the master node (D.17 and D.18 respectively) and worker node (D.19 and D.20 respectively). Currently, are implemented two non-spark analysers, one that gives some statistics about the captured network data (section 4.5.2) and the other that performs the analysis trying to find malicious activity (section 4.5.3) with both sharing the same application core (section 4.5.1).

4.5.1 Analyzer core

Every analysis application has something equivalent, and that is the application core. Its function is to get the raw data that is stored on a Kafka topic and extract the relevant information that will aid in the analysis.

The data is processed message by message. When the analyzer gets a message (also designated as chunk) from Kafka it starts by looping the chunk: first it extracts the packet header to know the exact length of the network packet and then it can retrieve the network packet. After the retrieval of the packet, the core parser will analyze it layer by layer until it reaches the last one. Since the network packets are captured from the ethernet layer (layer 2 of the OSI model) the parser knows how to start the extraction process. It will read which protocol is in the next layer; this way it can extract the information accordingly to the protocol specification (available on RFC documents). After it finishes extracting the information of the packet it will advance to the next until it reaches the end of the chunk; when that happens it will fetch another one if available; otherwise, it will wait for new data.

The core parser went trough three different versions. The first and second uses the Python frameworks Scapy and PyPacket, respectively. The last one uses a parser implemented from scratch (a custom parser). to decrease the time taken to parse the packets. Section 5.5 documents an experiment to compare the offline performance of these three parsers and on section 5.6 the performance overall system with the best parser is assessed.

4.5.2 Information analysis

This is one of the applications that run on the analysis cluster. Its functionality is as follows: it counts the number of Transmission Control Protocol (TCP), User Datagram Protocol (UDP), ARP and Internet Control Message Protocol (ICMP) packets and what is the source and destination IP address most frequent in the packets. Each X seconds (X is an integer value defined on the configuration file) the application will print the information described above to the console. Its source code can be found on appendix G, listing G.1. Listing 4.1, presents an example of the console output of the execution of this application.

Listing 4.1: Example of the console output produced by the information analyzer application

```
TCP packets: 375848, UDP packets: 769, ARP packets 30, ICMP packets 0
Top source IP: 10.0.0.10 with 313054 requests
Top destination IP: 10.0.0.11 with 313056 requests

```

```
TCP packets: 685274, UDP packets: 798, ARP packets 30, ICMP packets 0
Top source IP: 10.0.0.10 with 568343 requests
Top destination IP: 10.0.0.11 with 568345 requests


Packets analyzed: 686178
TCP packets: 685274, UDP packets: 798, ARP packets 30, ICMP packets 0
```

4.5.3 Malicious domain analysis

This is the other implemented analysis application, also coded on Python 3.9, that executes on the analysis cluster. It was taken the approach described in section 3.2; it is used a featureless machine learning model to detect malicious domains in near real-time. The stages of this application are as follows:

1. Analyze each message on the Kafka topic;

2. For each packet on the message, analyze the ones that match the DNS query packet signature (RFC 1035);

3. For the match packets extracts the domain name and remove the subdomains and the Top-level domain (TLD);

4. Query the machine learning with the extracted information and read the results;

5. Based on the result, check if the domain is benign or malicious and informs about it in the console.

The last stage (5) is simply for testing purposes and may be effortlessly adjusted. In a real scenario, the application should send a notification to the administrators or execute another application to analyze the entire traffic of the host that executed the query. An example of the console output of this application can be found on the listing 4.2.

Listing 4.2: Example of the console output produced by the malicious DGA analyzer application

```
Loading the model...
Load complete...

Request 10.1.2.155 -> 10.1.2.1: www.google.pt is benign with prob. of
↪ 94.56%
Request 10.1.2.155 -> 10.1.2.1: estig.ipb.pt is benign with prob. of
↪ 99.99%
Request 10.1.2.155 -> 10.1.2.1: hcbbfehgoqlw.ru is DGA with prob. of
↪ 99.98%
Request 10.1.2.155 -> 10.1.2.1: wykosev.com is DGA with prob. of 54.09%
Request 10.1.2.155 -> 10.1.2.1: fzcqvinskycattederifg.com is DGA with
↪ prob. of 100%
```

Dataset

The dataset (designated as the final mixed dataset) to train the machine learning model is constituted by two columns: the domain and the class columns. The values on the domain column are a string representing the domain name without the TLD and the subdomains. The values on the column class are either 0, meaning the domain is benign, or 1, meaning the domain is malicious.

Three different datasets were used, the Alexa, dataset containing only benign records; the Bambenek dataset, containing only malicious domains; the Splunk dataset, containing a mix of benign and malicious records. The number of records that each dataset contains can be viewed on table 4.5.

A set of steps are executed in order to generate the final dataset (overview on figure 4.4).

Table 4.5: Number of records per dataset

Dataset	Benign records	Malicious records	Total records
Alexa 1M	895,830	0	895,830
Bambenek DGA set	0	1,169,356	1,169,356
Splunk dataset	50,000	50,000	100,000
Final dataset	705,333	1,152,636	1,857,969

For the Alexa and Bambenek dataset, it is appended a new column for the identification of the domain' legitimacy: in the case of Alexa it will be appended a column with all the values set to 0, and on the Bambenek dataset with all the values set to 1. After this step, irrelevant columns will be dropped and they are ready to be joined. The Splunk dataset contains both benign and malicious domains that have the format of text "legi" or"dga"). In this case, instead of appending a new column, the values of the existing ones are changed to 0 when the value is"legit" or 1 when the value is"dga". After each dataset is properly formatted, they are joined together, originating a mixed dataset. To generate the final mixed dataset, two more steps are required: the striping of the domain name and the drop of duplicate rows. The dataset is a Pandas (Python framework) DataFrame object; iterating each row in this format takes some time; the process can be optimized by converting the dataset from the DataFrame object to a dictionary object; this allows a much faster iteration of the row to strip the domain name, even if in the end it's necessary to perform two conversions (DataFrame to dictionary and dictionary to DataFrame). After the dataset has the domains striped, the duplicated records are dropped and the entire dataset is flushed (reordering of the rows). The source code (based on `https://github.com/sudo-rushil/dga-intel-web`) of the dataset preparation for the machine learning malicious domain detection model can be found in appendix G, listing G.2.

Machine learning model

The model is built using the TensorFlow Python framework Keras API using the Long short-term memory (LSTM) neural network architecture (the model architecture is similar

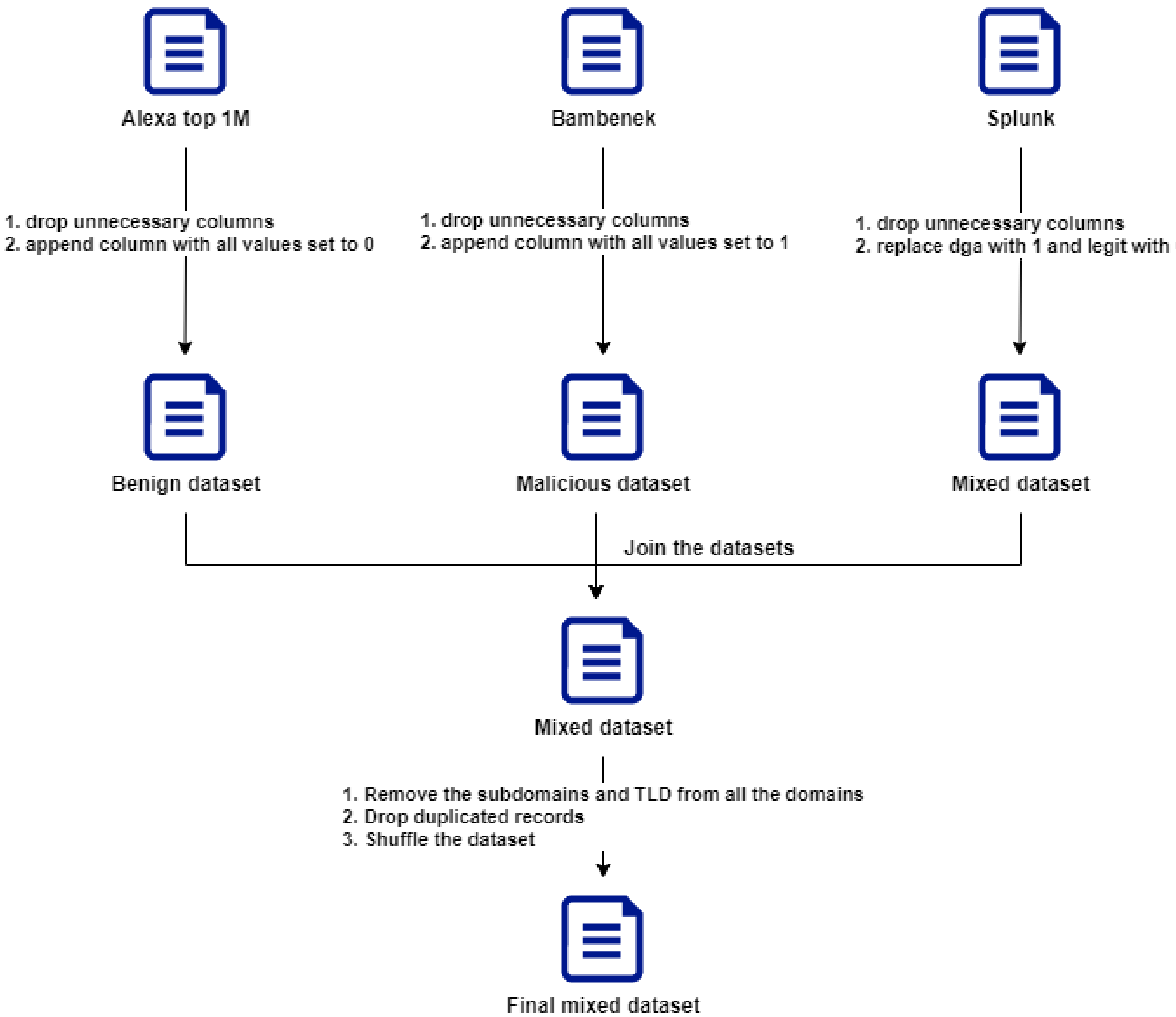

Figure 4.4: Steps to generate the final dataset to the malicious domains detection

to the one implemented by Yu B. et al [81]). As the domains names are a sequence of characters, LSTM was the model chosen as it can successfully process sequences.

The input is passed through a learnable embedding vector, with 39 of input dimension and 63 of input size, that converts each character into a 128-dimensional vector encoding its information. The number 39 comes from the possible characters that can appear on a domain name and the 63 corresponds to the max length of a domain name (without the TLD and subdomains, per the specification on RFC 1035). After passing through the embedding layer, it passes through a 1 dimension Convolutional Neural Network (CNN) with the ReLu activation function and then it runs through the LSTM layer with a dimension of 64 and is classified with a single dense layer using the SigMoid activation function.

The input of the model consists of a size 63 integer array (max length of a domain name without the subdomain and TLS); hence the string is converted to the array using a static mapping, where each character corresponds to a letter; in case the domain does not contain 63 characters, the rest of the array will be padded with zeros as the array must always be the size of 63 as it is the input size expected by the model.

From the final mixed dataset, the model used 90 % of the data as the training data and 10 % as the testing data and the training was conducted for 6 epochs. The results regarding the model can be found in section 5.7. The script that builds and trains the machine learning model can be found on appendix G listing G.3.

Chapter 5

Experiments and discussion

This chapter reports the experiments performed, explaining its objectives and discussing the results. Every experiment was conducted in the test-bed described in section 4.1. These experiments were performed against the maximum network load (1Gb/s) allowed on these hardware devices. This way it's possible to know how much the system can handle and how it will behave in the worst-case scenario (regarding the line rate). The devices on network A (the ones where the traffic will be captured) were communicating at 1Gb/s, which is the maximum bandwidth possible (the actual throughput during a 60 seconds communication can be found in figure 5.1). The tool to generate the network load between the two devices was the well-known Iperf3 application. To recap, it's designated by *full-packet capture* a network capture where each network packet is captured as is (minus the layer 1 of the OSI model) and by *headers-only capture*, a network capture where only the headers are captured (the OSI layer 1 is also not captured). In the following experiments the headers-only capture has the packets truncated at 96 bytes, allowing to acquire the data link, network and transport layers and also some bytes of the payload.

5.1 Packet capture

As described in section 4.2, before the final version of the packet capture application, the system had two others. As already stated the first version (version 1) opens TCPDump

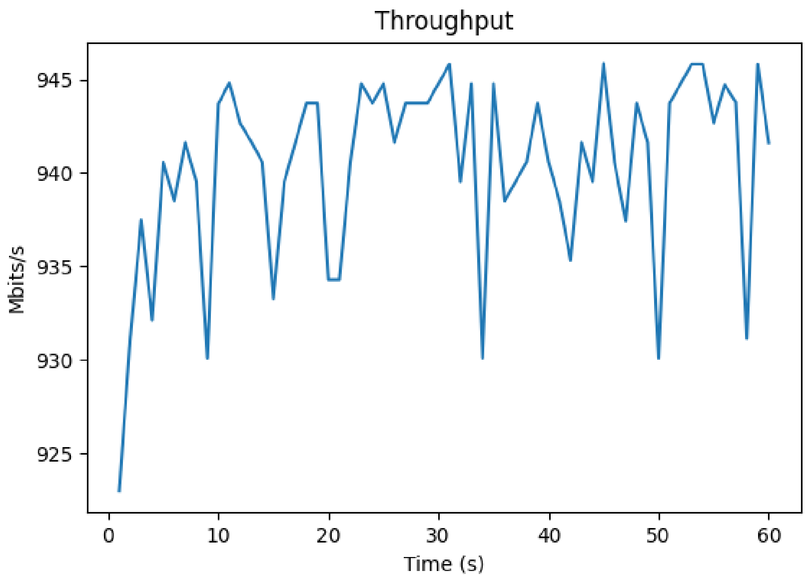

Figure 5.1: Network throughput between the two devices on network A during 60 seconds

as a subprocess and reads the network packets through a pipe; the second version (version 2) is implemented in C and uses a circular data buffer to temporarily store the network data; the final version (version 3) uses the local disk as a data buffer. These versions were compared in a first experiment.

The experiment was conducted with a Kafka topic containing one partition, with both full-packet and headers-only capture. The duration of the network capture, for each version was 60 seconds.

The comparison of the application versions regarding the number of captured and dropped packets can be visualized on table 5.1. Figure 5.2 is a graphical representation focused only on the packet loss.

Considering the results it's possible to conclude that a data buffer is mandatory to capture on a network with high bandwidth. Without it, the application can't handle all the load on a single thread as the publication of the message to the messaging queuing cluster takes more time than capturing the network packets. With the introduction of a data buffer, the application is now able to temporarily store the data on an efficient data

Table 5.1: Comparison of the three application versions, regarding the packet loss

	Version	Packets received	Packets captured	Packets dropped	Packet loss (%)
Headers only	1	4,860,868	778,435	4,071,554	84
	2	3,166,955	3,166,815	0	0
	3	4,980,088	4,979,413	0	0
Full packet	1	4,904,166	225,345	4,677,604	95.4
	2	3,944,170	2,608,671	1,335,499	33.86
	3	4,931,720	4,930,747	775	0.02

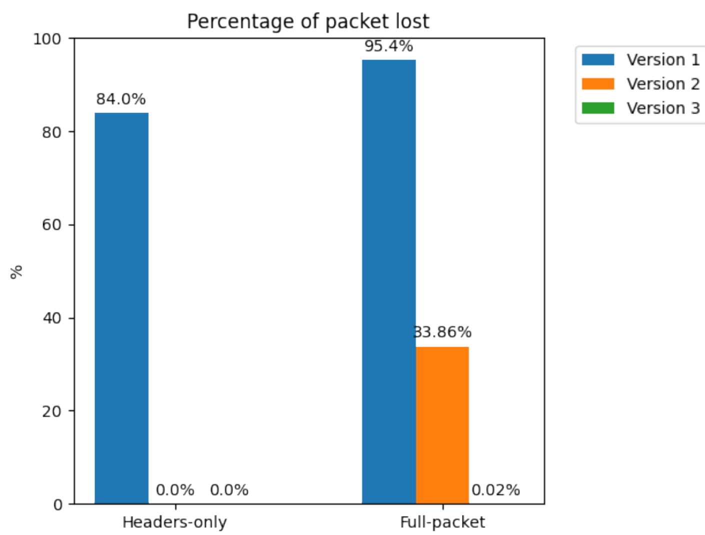

Figure 5.2: Comparison between the three packet capture applications regarding the packet loss

structure, this way the two threads (reader and writer) may work without waiting for one another. However, is not always possible to lock a large-sized data structure into the host's RAM and when the buffer is full, network data will start to be lost. Nevertheless, the application can now keep up with the load when performing the headers-only packet capture. With the data buffer stored on disk (as a file) the application can perform the full-packet capture although, this buffer is limited to the host's disk size and once the disk is full the operation will stop.
Each application version had an increase in performance being able to lose fewer packets. Future experiments will be executed using version 3 of the packet capture application.

5.2 Kafka message size

On a real-time system, every second counts and every procedure must be optimized to save the most time possible. As previewed in section 4.3.2, this experiment will answer the question of what is the Kafka message size that offers better performance.
The experiment was conducted using a network probe to capture network packets and produce the corresponding messages, and a host to perform the consumption (hosts specifications on section 4.1).
The parameters, under study is the message size with the following possible values:

- 131072 (2^{17}) bytes (128 KiB);
- 262144 (2^{18}) bytes (256 KiB);
- 524288 (2^{19}) bytes (512 KiB).

The output consists of 3 different values: upload delay, download delay and total delay (on seconds). The upload delay is the time between the capture of the last network packet to its publication on Kafka. Download delay is the time between the upload of the last message to the consumption of that last message. The total delay is the sum of the upload and download delay.

All the experiments were executed on a Kafka topic with two partitions, at full-packet and headers-only packet capture.

Figure 5.3 presents the results of a single 300 seconds headers-only capture. It's possible to conclude that any of the three message sizes have a very low delay, not differing too much from each other. Also, the delay does not grow and it's practically constant since on a 60 seconds capture the delays are essentially the same (figure 5.4). Now to determine the best message size a capture of only 60 seconds will not suffice since the size with lower delay changes even with the same experimental conditions, however, when performing a capture of 300 seconds the best message size is always 128 KiB, this conclusion emerged by performing five consecutive experiments whose the aggregated results can be consulted on table 5.2. These results demonstrate that the size of 128KiB offers the lowest value on the average of the results also, the standard deviation is low meaning that the values do not differ too much from each experiment.

Table 5.2: Aggregation of five headers-only 300 seconds packet capture experiments

	Average delay (seconds)			Standard deviation (seconds)		
Message size (KiB)	Upload	Download	Total	Upload	Download	Total
128	0.32	1.02	1.34	0.16	0.02	0.16
256	0.81	1.02	1.83	0.15	0.01	0.16
512	0.78	0.96	1.74	0.17	0.12	0.21

Figure 5.5 presents the results of a single 300 seconds full-packet capture. After analyzing the chart, one can conclude that producing messages to Kafka in chunks of size 512 KiB gives a better performance both on the upload and on the download. To better enforce this conclusion it was performed five consecutive experiments whose the aggregated results can be consulted on table 5.3. The size of 512 KiB offers the lowest average and the best standard deviation, while on the other sizes the deviation is higher, meaning that the system is not so stable.

Since there isn't much difference in performance from the message size in the header-only packet capture and the best size for the full-capture is 512 KiB, for future experiments it will be used the size of 512 KiB for the messages.

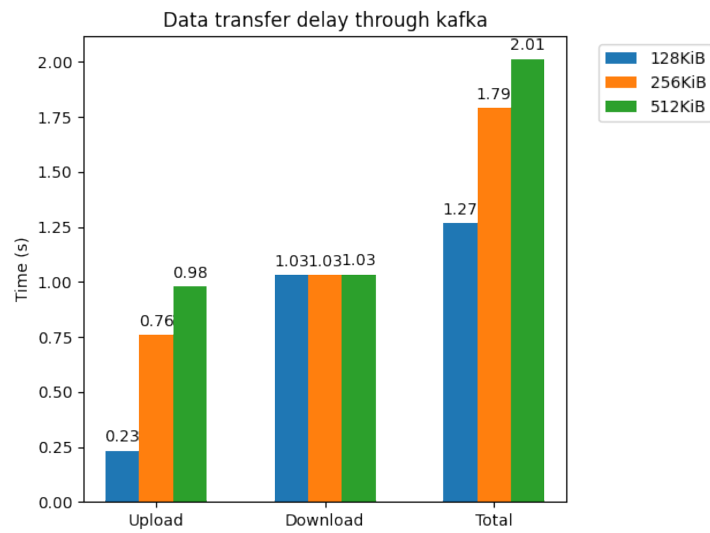

Figure 5.3: Kafka consumer and producer delay in a headers-only packet capture at 1Gb/s during a 300 seconds capture

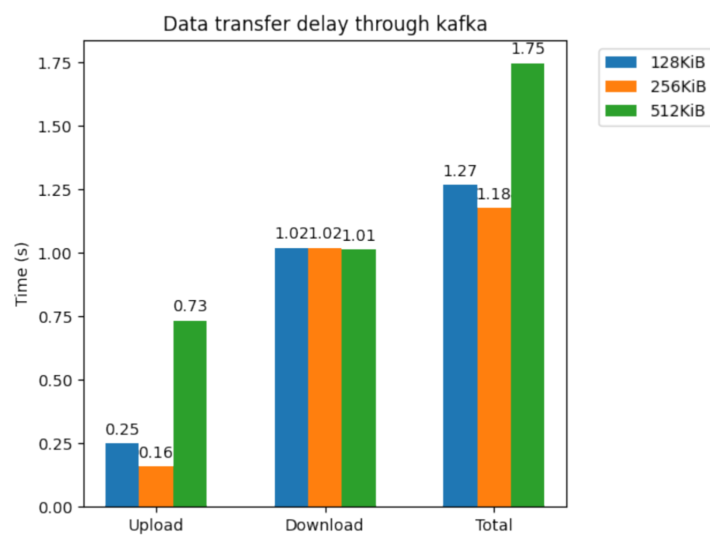

Figure 5.4: Kafka consumer and producer delay on a headers-only packet capture at 1Gb/s during 60 seconds

Table 5.3: Aggregation of five full-packet 300 seconds packet capture experiments

	Average delay (seconds)			Standard deviation (seconds)		
Message size (KiB)	Upload	Download	Total	Upload	Download	Total
128	82,60	21.19	103.79	19.25	5.42	15.21
256	82.43	69.55	151.98	24.53	10.08	29.14
512	65.66	1.21	66.88	1.82	0.32	2.06

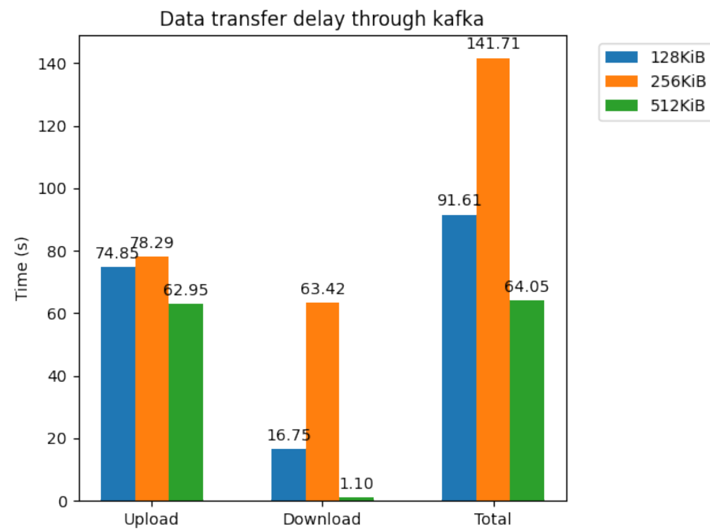

Figure 5.5: Kafka consumer and producer delay in full-packet capture at 1Gb/s during a 300 seconds capture

5.3 Kafka partitions

This experiment was executed to answer the questions from section 4.3.2. The experiment was performed in a duration of 60 seconds while publishing the messages to Kafka in chunks of 512 KiB. To note that in this experiment every necessary Kafka node had only one partition assigned.

After the analysis of the results (figure 5.6) we can see a tremendous improvement when working with more than a single partition. We can also see that working with three partitions is better than working with four. The ideal should be to allocate three partitions per topic but to mention that the number of nodes in a Kafka cluster is limited so, the number of partitions to allocate should be considered in a case-to-case scenario as there isn't any improvement in having multiple partitions if they are allocated on the same Kafka node.

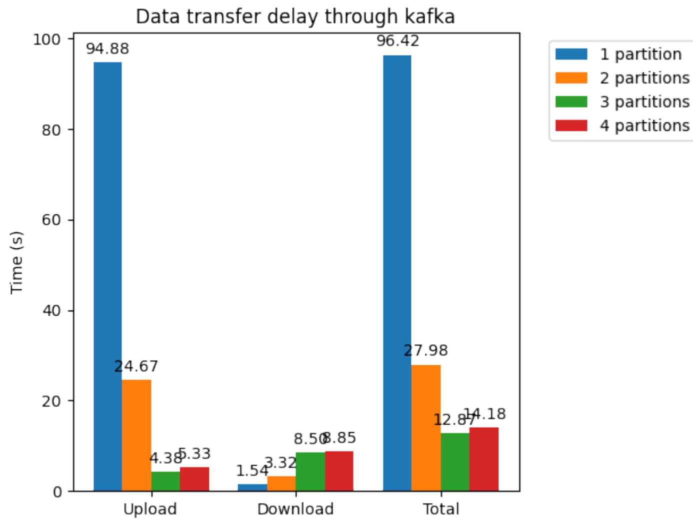

Figure 5.6: Consumer and producer delay on a 60 seconds 1Gb/s full-packet capture regarding a different number of Kafka partitions assigned

5.4 HDFS performance

This experiment has the objective to know the system performance when the system is capturing and storing the file persistently into the HDFS cluster. A comparison of the performance when encryption is enabled between the clients and the nodes will also be performed.

The experiments were executed with a Kafka topic containing two partitions, the messages were published to Kafka in chunks of 512 KiB, two HDFS nodes available and the replica of the distributed file was set to two, meaning that the file will be replicated in two nodes. The upload delay is the time between the capture of the last network packet to its publication on Kafka. Download delay is the time between the upload of the last message to the storage of that last message on the distributed file located on the HDFS cluster. The total delay is the sum of the upload and download delay.

Figures 5.7 and 5.9 shows very low delay between the capture (headers-only) and the insertion of the file on HDFS and can be consider a near real-time operation. Regarding the security, even if working with plain text (without encryption between the client and the nodes) has better performance, using it would compromise the security requirements of this system (described in section 2.4).

On figures 5.8 and 5.10 it is represented the delay when performing and saving a full-packet capture.

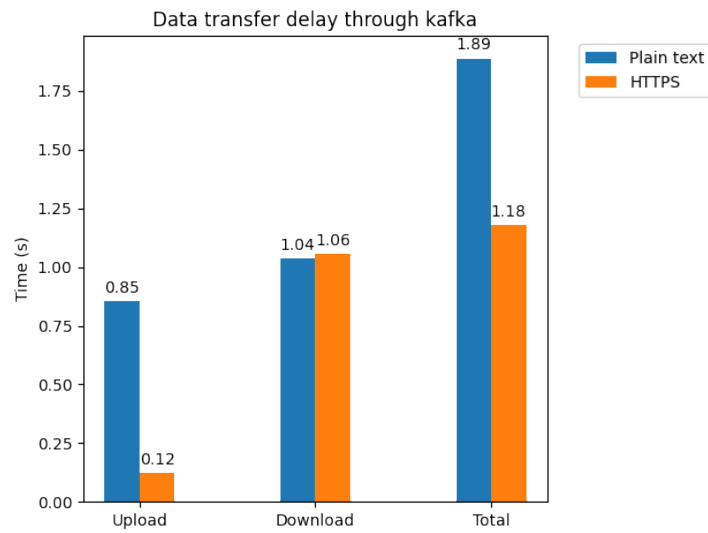

Figure 5.7: HDFS consumer and producer delay in a headers-only packet capture at 1Gb/s during 60 seconds

As this module purpose is to store (and retrieve when necessary) the network data persistently, there isn't an issue if the operation doesn't occur in near real-time as long as it consumers all messages from Kafka without losing any.

5.5 Core parser stress test

This experiment was conducted in offline mode, meaning that instead of performing the experiment connected to a live capturing system, it instead was executed reading a local file, thus producing maximum bound results. The experiment was conducted with two PCAP files. One of the files contains a full-packet capture (file A) carrying 5.708.800 packets with a total size of 10 GB. The other file (file B) contains a header-only (packets truncated at 96 bytes) packet capture carrying 102.152.015 packets with a total size of also 10 GB.

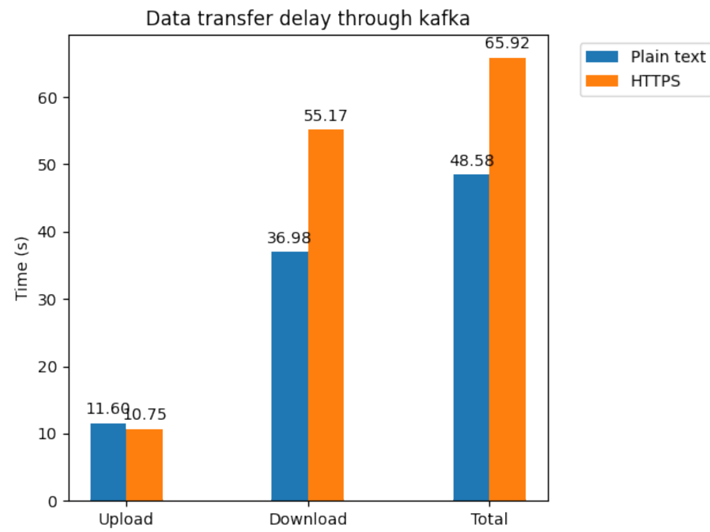

Figure 5.8: HDFS consumer and producer delay in full-packet capture at 1Gb/s during 60 seconds

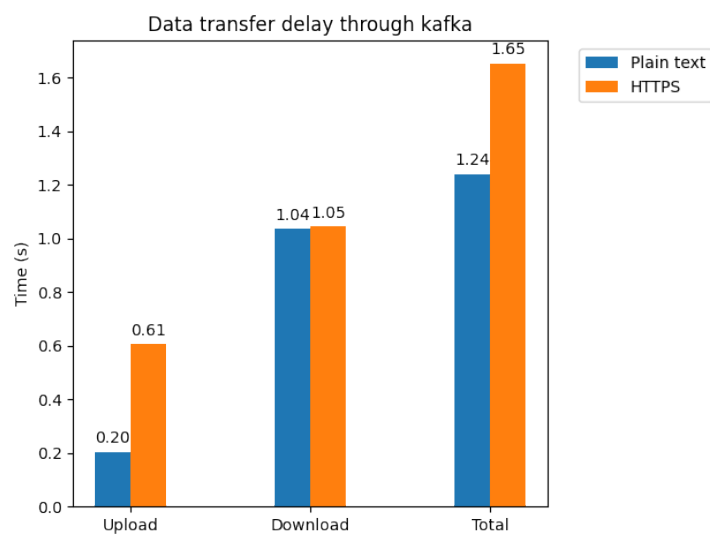

Figure 5.9: HDFS consumer and producer delay in a headers-only packet capture at 1Gb/s during 300 seconds

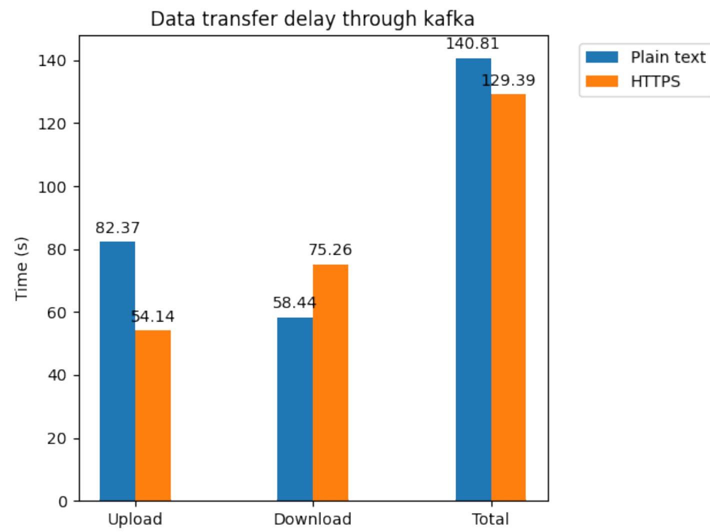

Figure 5.10: HDFS consumer and producer delay in full-packet capture at 1Gb/s during 300 seconds

The results of this experiment show which of the three parsers is faster and the average of the processed packets per second. With figures 5.11 and 5.13 it is possible to conclude that the custom parser is faster on processing both PCAP files. Also, on figures 5.12 and 5.14 it is presented the maximum of packets that each parser can process per second. The custom parser has the best performance as it only parses the necessary information, while the other two parse the entire packet, wasting computation resources and time extracting information that is irrelevant to the analyzer. For these reasons the custom parser is the one selected to be deployed within the core.

5.6 Core parser performance

While on section 5.5 were performed experiments on offline mode to find the fastest parser for the core, now it's time to find out if that parser can analyze the network data in near real-time.

For this intent, two experiments were conducted one while capturing the headers-only and the other while capturing the full packet, both with a duration of 300 seconds.

Figures 5.15 and 5.16 present the results of headers the only and full packet capture,

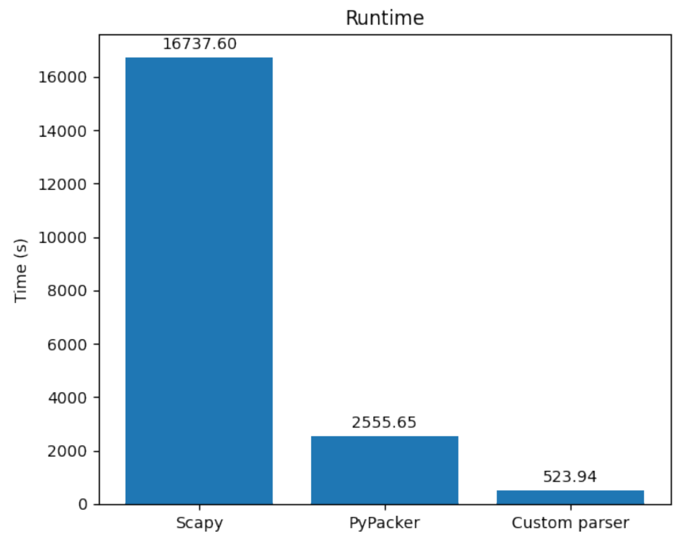

Figure 5.11: Runtime of the different core parsers when processing file B

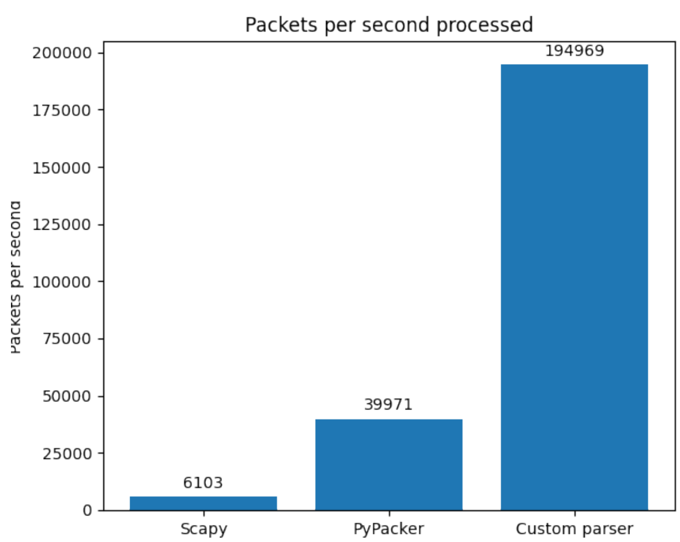

Figure 5.12: Average of the processed packets per second of the different core parsers when processing file B

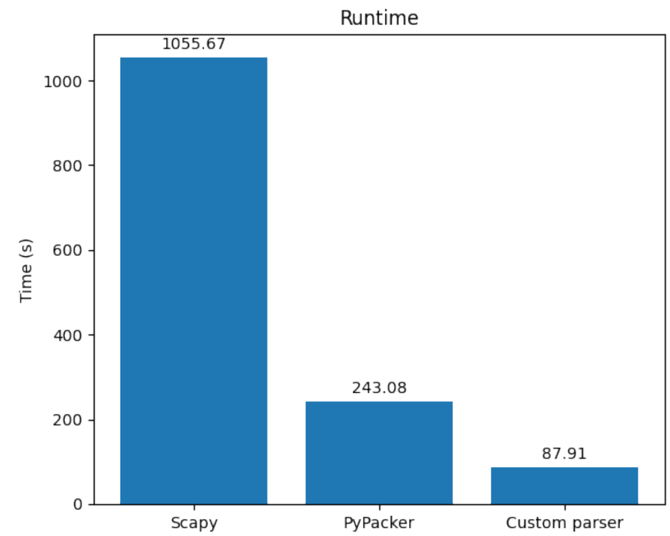

Figure 5.13: Runtime of the different core parsers when processing file A

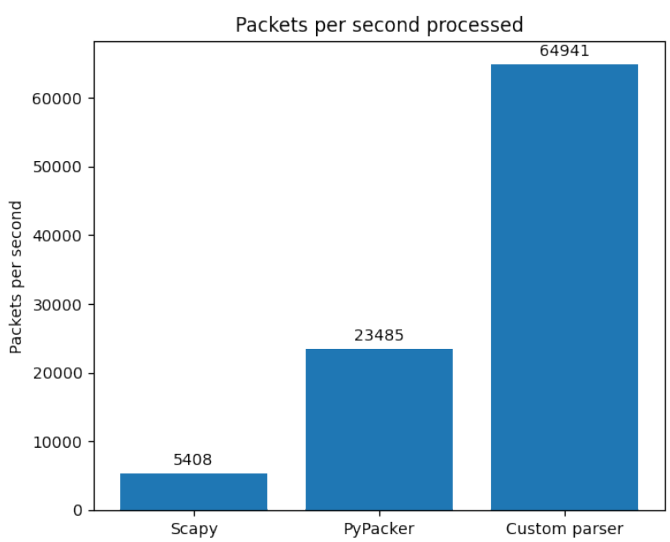

Figure 5.14: Average of the processed packets per second of the different core parsers when processing file A

respectively. After analysing the data, one can say that while capturing only the headers the system can analyse the data in near real-time with very low delay. In turn, while capturing at full packet the analyzer can still keep up with the rate and have a very low delay analyzing the packets, being the majority of the delay due to the upload from the network probe to Kafka. Nevertheless, in the last situation the system took barely more than 60 seconds to complete the task and such is still a good result to prevent some network attacks. But it should be noted that this performance test is only about the core parser (where it extracts relevant information), and there isn't any kind of processing of the extracted data.

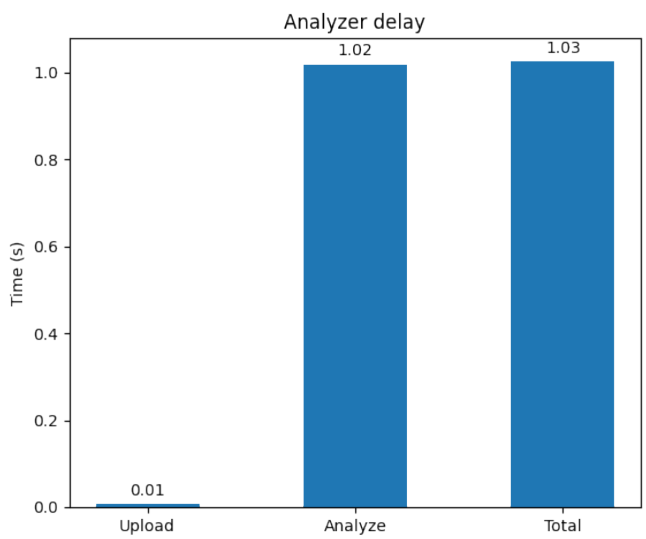

Figure 5.15: Performance of the core parser while performing a headers-only packet capture

5.7 Machine learning model

The training of the model was performed in one of the analysis nodes taking about 7 minutes per epoch (6 in total) with a total of 42 minutes. It was used 10,000 records from the testing dataset to evaluate the model relating the precision and the recall.

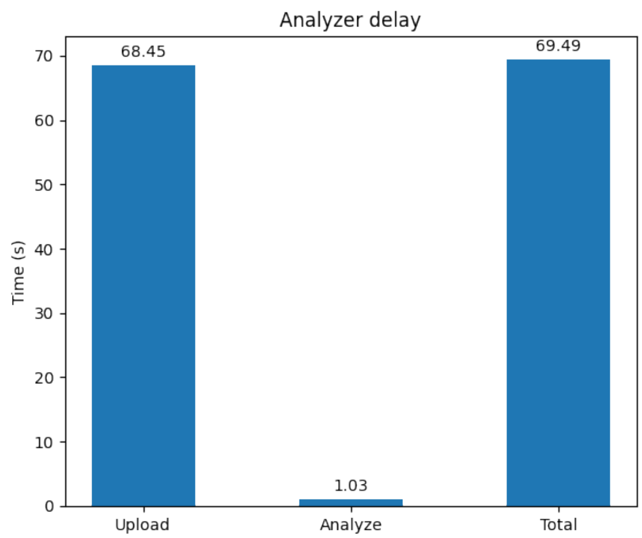

Figure 5.16: Performance of the core parser while performing a full-packet capture

Figure 5.17 show the confusion matrix describing the number of true positives, false positives, true negatives and false negatives. A true positive (TP) value is when the model classifies the value correctly as positive, a false positive (FP) is when the model classifies as positive but the actual result is negative, a true negative (TN) is when the model classifies the value correctly as negative and a false negative (FN) is when the model classifies as negative but the result is in fact positive.

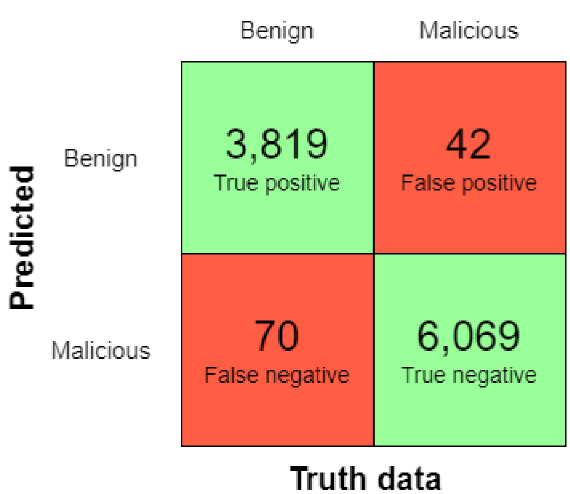

Figure 5.17: Confusion matrix of the domain classifier of the machine learning model

The precision of a model tells the percentage of the classifications made by the model as being correct. The recall tells how many of the positive cases were correctly classified by the model.

The precision P given the number of true positives TP and the false positives FP can be calculated by the formula 5.1:

$$P = \frac{TP}{TP + FP} \tag{5.1}$$

The recall R given the number of true positive TP and the false negative FN can be calculated by the formula 5.2:

$$R = \frac{TP}{TP + FN} \tag{5.2}$$

The precision and recall of this model are 98.9 % and 98.2 % respectively, meaning that it's a solid model that can be deployed in a real scenario. Regarding the evaluation results, the model classified 42 samples incorrectly as positive meaning that, in case the system automatically drops the connection when it detects a malicious domain, it may block important communications. Therefore it's important to precisely decide what to do when the system classifies domains as malicious. In the case of false negatives, the connection transits without alarming the system, and that may be an issue since it allows malicious communications.

New DGA algorithms, that generate new malicious domains that are harder to detect, are always appearing. So, when a new algorithm is discovered, or when a new family of malicious domains are captured, they should be appended to the training set and the machine learning module should be retrained. This way, the system has always the best possible detection rate. Nevertheless, not training the module with the newest data does not mean that a new family of malicious domains are not detected; they still may be and that is the usefulness of using machine learning to perform these kinds of detections. Now, the module shouldn't be trained in production as such takes a long period; therefore the model should be trained outside the system and then the updated trained module should be inserted into the system (stored on the HDFS cluster) so the analysers may update theirs. Anyway, the detection rate is not 100 %. Consequently, to further improve the malicious detection rate, the system needs to have multiple different analysis services.

Chapter 6

Conclusions and future work

This work, developed within the “CybersSEC IP - CYBERSecurity SciEntific Competences and Innovation Potential (NORTE-01-0145-FEDER-000044)” research project, described and implemented an approach for network data capturing and analysis. The conceptualization is based on relevant scientific literature, assessing the knowledge that emerges from them, allowing to frame the system in contemporary IT infrastructures and requirements. With this in mind, a flexible, scalable and practical architecture is implemented, keeping a low impact on the network.

With the support of the experiment’s results, it is safe to say that the system can capture and analyze network data in near real-time when performing a headers-only packet capture, generating a low computational load on the system. Regarding the full packet capture when the maximum throughput is achieved, the system presents some delay that may put at risk the requirement of near real-time. Nevertheless all the experiments were performed in the worst-case scenario to stress the system to the fullest; in a real case situation that won’t happen all the time.

The scalability and modularity are also assured: if additional analysis tools (modules) are needed, these can be easily implemented, deployed and executed in parallel without modifying the current system or other running applications, while all of them may use the same information. During the implementation of this system, it wasn’t implemented a Spark analyzer, although the system supports and is ready to receive them.

Concerning one of the analysis modules, the detection of malicious domains, the system has a recall of 98 % being a reliable detection model. Besides the good accuracy it's not perfect as some of the malicious domains may not be detected. That said, to further increase the detection of malicious activity, the system can and should have more than one analyzer to detect malicious activity as it is prepared to run multiple modules on the analyzer sub-system at the same time.
The proposed objectives of this project were fulfilled with intended results, the system is able to capture and analyse network data in near real-time.

6.1 Future Work

Besides the work objectives being fulfilled the project is far from being completed. There are still many pathways to explore, namely the following. Start to analyze the traffic in a real network with 1Gb/s of maximum bandwidth and without generating the network traffic. Instantiate the architecture in a test scenario, composed of IoT devices, regular workstations, etc. to allow the assessment of the scalability and flexibility of the system. Implement a connection flow analysis application in order to assess the system about the functionality and performance on reordering the network data before performing analysis (this way another set of malicious activity can be identified). Implement more analysis applications to increase the chance of catching malicious activity. Improve the analysis application so it may generate reports and apply certain automatic actions, like requesting the hardware to block certain IP addresses (this way the system may also be an IPS). Automatically select which data tier to capture (header-only or full-packet) based on the type of attack, since not all of them need the full-packet information to be detected. It is also planned to create and publish two more articles based on the presented work. The first one, is almost finished and is related with the performance of the packet capture and the other will be regarding the results of the analyzers.

Bibliography

[1] G. Coulouris, *Distributed systems : concepts and design*. Boston: Addison-Wesley, 2012, ISBN: 978-1299999633.

[2] M. Efatmaneshnik, S. Shoval and L. Qiao, 'A standard description of the terms module and modularity for systems engineering,' *IEEE Transactions on Engineering Management*, vol. 67, no. 2, pp. 365–375, 2020. DOI: `10.1109/TEM.2018.2878589`.

[3] I. A. Ibrahim Diyeb, A. Saif and N. A. Al-Shaibany, 'Ethical Network Surveillance using Packet Sniffing Tools: A Comparative Study,' *International Journal of Computer Network and Information Security*, vol. 10, no. 7, pp. 12–22, Jul. 2018, ISSN: 20749090. DOI: `10.5815/ijcnis.2018.07.02`.

[4] E. Luke, 'Defining and measuring scalability,' in *Proceedings of Scalable Parallel Libraries Conference*, 1993, pp. 183–186. DOI: `10.1109/SPLC.1993.365568`.

[5] M. Abercrombie, *Is personal data the new gold?* 2020. [Online]. Available: `https://www.libf.ac.uk/news-and-insights/news/detail/2020/05/22/is-personal-data-the-new-gold` (visited on 09/06/2021).

[6] Lionel Sujay Vailsher, *Global IoT and non-IoT connections 2010-2025*, 2021. [Online]. Available: `https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/` (visited on 13/05/2021).

[7] Worldometer, *World Population Projections - Worldometer*, 2021. [Online]. Available: `https://www.worldometers.info/world-population/world-population-projections/` (visited on 13/05/2021).

[8] S. Lysenko, K. Bobrovnikova, R. Shchuka and O. Savenko, 'A cyberattacks detection technique based on evolutionary algorithms,' in *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, 2020, pp. 127–132. DOI: 10.1109/DESSERT50317.2020.9125016.

[9] R. Johnson, *60 Percent of Small Companies Close Within 6 Months of Being Hacked*, 2019. [Online]. Available: https://cybersecurityventures.com/60-percent-of-small-companies-close-within-6-months-of-being-hacked/ (visited on 13/05/2021).

[10] T. Akolawala, *Data of Over 92 Percent LinkedIn Users Exposed in New Breach: Report*, Jun. 2021. [Online]. Available: https://gadgets.ndtv.com/apps/news/linkedin-data-breach-hack-700-million-92-percent-users-personal-information-sold-online-report-2475268.

[11] E. Kent, *CD Projekt hit by "targeted cyber attack"*, 2021. [Online]. Available: https://www.eurogamer.net/articles/2021-02-09-cd-projekt-hit-by-targeted-cyber-attack (visited on 13/05/2021).

[12] Stempel Jonathan and Finkle Jim, *Yahoo says all three billion accounts hacked in 2013 data theft | Reuters*, 2017. [Online]. Available: https://www.reuters.com/article/us-yahoo-cyber/yahoo-says-all-three-billion-accounts-hacked-in-2013-data-theft-idUSKCN1C8201 (visited on 13/05/2021).

[13] Dan Swinhoe, *The 15 biggest data breaches of the 21st century | CSO Online*, 2021. [Online]. Available: https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html (visited on 13/05/2021).

[14] P. Roquero, E. Magaña, R. Leira and J. Aracil, 'Performance evaluation of client-based traffic sniffing for very large populations,' *Computer Networks*, vol. 166, p. 106 985, Jan. 2020, ISSN: 13891286. DOI: 10.1016/j.comnet.2019.106985.

[15] S. H. Mousavi, M. Khansari and R. Rahmani, 'A fully scalable big data framework for Botnet detection based on network traffic analysis,' *Information Sciences*,

vol. 512, pp. 629–640, Feb. 2020, ISSN: 00200255. DOI: 10.1016/j.ins.2019.10.018.

[16] P. Emmerich, M. Pudelko, S. Gallenmüller and G. Carle, 'FlowScope: Efficient packet capture and storage in 100 Gbit/s networks,' in *2017 IFIP Networking Conference, IFIP Networking 2017 and Workshops*, vol. 2018-Janua, Institute of Electrical and Electronics Engineers Inc., Jul. 2017, pp. 1–9, ISBN: 9783901882944. DOI: 10.23919/IFIPNetworking.2017.8264852.

[17] J. Uramova, P. Segec, M. Moravcik, J. Papan, T. Mokos and M. Brodec, 'Packet capture infrastructure based on Moloch,' in *ICETA 2017 - 15th IEEE International Conference on Emerging eLearning Technologies and Applications, Proceedings*, Institute of Electrical and Electronics Engineers Inc., Nov. 2017, ISBN: 9781538632963. DOI: 10.1109/ICETA.2017.8102538.

[18] M. Z. N. L. Saavedra and W. E. Yu, 'Towards large scale packet capture and network flow analysis on hadoop,' in *Proceedings - 2018 6th International Symposium on Computing and Networking Workshops, CANDARW 2018*, Institute of Electrical and Electronics Engineers Inc., Dec. 2018, pp. 186–189, ISBN: 9781538691847. DOI: 10.1109/CANDARW.2018.00043.

[19] M. T. Tun, D. E. Nyaung and M. P. Phyu, 'Performance Evaluation of Intrusion Detection Streaming Transactions Using Apache Kafka and Spark Streaming,' in *2019 International Conference on Advanced Information Technologies, ICAIT 2019*, Institute of Electrical and Electronics Engineers Inc., Nov. 2019, pp. 25–30, ISBN: 9781728151731. DOI: 10.1109/AITC.2019.8920960.

[20] A. M. Karimi, Q. Niyaz, Weiqing Sun, A. Y. Javaid and V. K. Devabhaktuni, 'Distributed network traffic feature extraction for a real-time IDS,' in *2016 IEEE International Conference on Electro Information Technology (EIT)*, vol. 2016-Augus, IEEE, May 2016, pp. 0522–0526, ISBN: 978-1-4673-9985-2. DOI: 10.1109/EIT.2016.7535295.

[21] F. L. Aryeh, B. K. Alese and O. Olasehinde, 'Graphical analysis of captured network packets for detection of suspicious network nodes,' in *2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment, Cyber SA 2020*, Institute of Electrical and Electronics Engineers Inc., Jun. 2020, ISBN: 9781728166902. DOI: 10.1109/CyberSA49311.2020.9139672.

[22] E. H. Do and V. N. Gadepally, 'Classifying Anomalies for Network Security,' in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2020-May, Institute of Electrical and Electronics Engineers Inc., May 2020, pp. 2907–2911, ISBN: 9781509066315.

[23] Y. T. Guo, Y. Gao, Y. Wang, M. Y. Qin, Y. J. Pu, Z. Wang, D. D. Liu, X. J. Chen, T. F. Gao, T. T. Lv and Z. C. Fu, 'DPI & DFI: A Malicious Behavior Detection Method Combining Deep Packet Inspection and Deep Flow Inspection,' in *Procedia Engineering*, vol. 174, Elsevier Ltd, Jan. 2017, pp. 1309–1314. DOI: 10.1016/j.proeng.2017.01.276.

[24] A. Ulmer, D. Sessler and J. Kohlhammer, 'NetCapVis: Web-based progressive visual analytics for network packet captures,' in *2019 IEEE Symposium on Visualization for Cyber Security, VizSec 2019*, Institute of Electrical and Electronics Engineers Inc., Oct. 2019, ISBN: 9781728138763. DOI: 10.1109/VizSec48167.2019.9161633.

[25] X. Ye, S. Qiao, N. Han, K. Yue, T. Wu, L. Yang, F. Huang and C.-a. Yuan, 'Algorithm for detecting anomalous hosts based on group activity evolution,' *Knowledge-Based Systems*, vol. 214, p. 106 734, Feb. 2021, ISSN: 09507051. DOI: 10.1016/j.knosys.2020.106734.

[26] H. Ichise, Y. Jin, K. Iida and Y. Takai, 'Detection and Blocking of Anomaly DNS Traffic by Analyzing Achieved NS Record History,' in *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, IEEE, Nov. 2018, pp. 1586–1590, ISBN: 978-9-8814-7685-2. DOI: 10.23919/APSIPA.2018.8659739.

[27] E. Longo, A. E. Redondi and M. Cesana, 'Accurate occupancy estimation with WiFi and bluetooth/BLE packet capture,' *Computer Networks*, vol. 163, p. 106 876, Nov. 2019, ISSN: 13891286. DOI: 10.1016/j.comnet.2019.106876.

[28] A. Lara and B. Ramamurthy, 'OpenSec: Policy-Based Security Using Software-Defined Networking,' *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 30–42, Mar. 2016, ISSN: 1932-4537. DOI: 10.1109/TNSM.2016.2517407.

[29] A. Subahi and G. Theodorakopoulos, 'Ensuring Compliance of IoT Devices with Their Privacy Policy Agreement,' in *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, IEEE, Aug. 2018, pp. 100–107, ISBN: 978-1-5386-7503-8. DOI: 10.1109/FiCloud.2018.00022.

[30] P. Boonyopakorn, 'Applying Data Analytics to Findings of User Behaviour Usage in Network Systems,' in *2018 International Conference on Information Technology (InCIT)*, IEEE, Oct. 2018, pp. 1–6. DOI: 10.23919/INCIT.2018.8584865.

[31] 'A Survey on Network Security-Related Data Collection Technologies,' *IEEE Access*, vol. 6, pp. 18 345–18 365, Mar. 2018, ISSN: 21693536. DOI: 10.1109/ACCESS.2018.2817921.

[32] J.-P. A. Yaacoub, H. N. Noura, O. Salman and A. Chehab, 'Digital Forensics vs. Anti-Digital Forensics: Techniques, Limitations and Recommendations,' Mar. 2021. arXiv: 2103.17028. [Online]. Available: https://arxiv.org/abs/2103.17028v1%20http://arxiv.org/abs/2103.17028.

[33] Y. Choi, J.-Y. Lee, S. Choi, J.-H. Kim and I. Kim, 'Traffic storing and related information generation system for cyber attack analysis,' in *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, IEEE, Oct. 2016, pp. 1052–1057, ISBN: 978-1-5090-1325-8. DOI: 10.1109/ICTC.2016.7763366. [Online]. Available: http://ieeexplore.ieee.org/document/7763366/.

[34] A. Horneman and N. Dell, 'Smart Collection and Storage Method for Network Traffic Data,' Sep. 2014.

[35] D. Zhou, Z. Yan, Y. Fu and Z. Yao, *A survey on network data collection*, Aug. 2018. DOI: 10.1016/j.jnca.2018.05.004.

[36] J. Quittek, T. Zseby, B. Claise and S. Zander, *Requirements for IP Flow Information Export (IPFIX)*, 2004. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3917 (visited on 13/05/2021).

[37] J. Parry, D. Hunter, K. Radke and C. Fidge, 'A network forensics tool for precise data packet capture and replay in cyber-physical systems,' in *Proceedings of the Australasian Computer Science Week Multiconference*, vol. 01-05-Febr, New York, NY, USA: ACM, Feb. 2016, pp. 1–10, ISBN: 9781450340427. DOI: 10.1145/2843043.2843047. [Online]. Available: http://dx.doi.org/10.1145/2843043.2843047%20https://dl.acm.org/doi/10.1145/2843043.2843047.

[38] G. Harris and M. C. Richardson, *Pcap capture file format*. [Online]. Available: https://tools.ietf.org/id/draft-gharris-opsawg-pcap-00.html (visited on 10/10/2021).

[39] A. H. Medalla, M. Z. N. L. Saavedra, P. A. R. Abu and W. E. S. Yu, 'Adapting block-sized captures for faster network flow analysis on the hadoop ecosystem,' in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, 2018, pp. 1097–1103. DOI: 10.1109/CompComm.2018.8780880.

[40] N. J. Venkatesan, E. Kim and D. R. Shin, 'PoN: Open source solution for real-time data analysis,' in *2016 3rd International Conference on Digital Information Processing, Data Mining, and Wireless Communications, DIPDMWC 2016*, Institute of Electrical and Electronics Engineers Inc., Aug. 2016, pp. 313–318, ISBN: 9781467393799. DOI: 10.1109/DIPDMWC.2016.7529409.

[41] M. Wahal, T. Choudhury and M. Arora, 'Intrusion Detection System in Python,' in *Proceedings of the 8th International Conference Confluence 2018 on Cloud Computing, Data Science and Engineering, Confluence 2018*, Institute of Electrical and Electronics Engineers Inc., Aug. 2018, pp. 348–353, ISBN: 9781538617182. DOI: 10.1109/CONFLUENCE.2018.8442909.

[42] D. Álvarez Robles, P. Nuño, F. González Bulnes and J. C. Granda Candás, 'Performance analysis of packet sniffing techniques applied to network monitoring,' *IEEE Latin America Transactions*, vol. 19, no. 3, pp. 490–499, 2021. DOI: 10.1109/TLA.2021.9447699.

[43] J. H. Moon and Y. T. Shine, 'A study of distributed SDN controller based on apache kafka,' in *Proceedings - 2020 IEEE International Conference on Big Data and Smart Computing, BigComp 2020*, Institute of Electrical and Electronics Engineers Inc., Feb. 2020, pp. 44–47, ISBN: 9781728160344. DOI: 10.1109/BigComp48618.2020.0-101.

[44] K. Team, *Documentation.* [Online]. Available: https://kafka.apache.org/documentation (visited on 20/09/2021).

[45] C. N. Nguyen, J.-S. Kim and S. Hwang, 'Koha: Building a kafka-based distributed queue system on the fly in a hadoop cluster,' in *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, 2016, pp. 48–53. DOI: 10.1109/FAS-W.2016.23.

[46] S. Kumar and E. Goel, 'Changing the world of Autonomous Vehicles using Cloud and Big Data,' in *Proceedings of the International Conference on Inventive Communication and Computational Technologies, ICICCT 2018*, Institute of Electrical and Electronics Engineers Inc., Sep. 2018, pp. 368–376, ISBN: 9781538619742. DOI: 10.1109/ICICCT.2018.8473347.

[47] D. Surekha, G. Swamy and S. Venkatramaphanikumar, 'Real time streaming data storage and processing using storm and analytics with Hive,' in *Proceedings of*

2016 International Conference on Advanced Communication Control and Computing Technologies, ICACCCT 2016, Institute of Electrical and Electronics Engineers Inc., Jan. 2017, pp. 606–610, ISBN: 9781467395458. DOI: 10.1109/ICACCCT.2016.7831712.

[48] A. Toshniwal, K. S. Rathore, A. Dubey, P. Dhasal and R. Maheshwari, 'Media streaming in cloud with special reference to amazon web services: A comprehensive review,' in *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*, 2020, pp. 368–372. DOI: 10.1109/ICICCS48265.2020.9121097.

[49] S. Ghemawat, H. Gobioff and S.-T. Leung, 'The google file system,' in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, 2003, pp. 20–43.

[50] B. P. Rao and N. N. Rao, 'HDFS memory usage analysis,' in *Proceedings of the International Conference on Inventive Computing and Informatics, ICICI 2017*, Institute of Electrical and Electronics Engineers Inc., May 2018, pp. 1041–1046, ISBN: 9781538640319. DOI: 10.1109/ICICI.2017.8365298.

[51] Y. Tian and X. Yu, 'Trustworthiness study of HDFS data storage based on trustworthiness metrics and KMS encryption,' in *Proceedings of 2021 IEEE International Conference on Power Electronics, Computer Applications, ICPECA 2021*, Institute of Electrical and Electronics Engineers Inc., Jan. 2021, pp. 962–966, ISBN: 9781728190037. DOI: 10.1109/ICPECA51329.2021.9362537.

[52] K. S. Madhu, B. C. Reddy, C. H. Damarukanadhan, M. Polireddy and N. Ravinder, 'Real Time Sentimental Analysis on Twitter,' in *Proceedings of the 6th International Conference on Inventive Computation Technologies, ICICT 2021*, Institute of Electrical and Electronics Engineers Inc., Jan. 2021, pp. 1030–1034, ISBN: 9781728185019. DOI: 10.1109/ICICT50816.2021.9358772.

[53] S. Mishra, P. K. Shukla and R. Agarwal, 'Location wise opinion mining of real time twitter data using hadoop to reduce cyber crimes,' in *2nd International Conference*

on Data, Engineering and Applications, IDEA 2020, Institute of Electrical and Electronics Engineers Inc., Feb. 2020, ISBN: 9781728157184. DOI: 10.1109/IDEA49133.2020.9170700.

[54] K. Aziz, D. Zaidouni and M. Bellafkih, 'Real-time data analysis using Spark and Hadoop,' in *Proceedings of the 2018 International Conference on Optimization and Applications, ICOA 2018*, Institute of Electrical and Electronics Engineers Inc., May 2018, pp. 1–6, ISBN: 9781538642252. DOI: 10.1109/ICOA.2018.8370593.

[55] R. Kamal, M. A. Shah, A. Hanif and J. Ahmad, 'Real-time opinion mining of Twitter data using spring XD and Hadoop,' in *ICAC 2017 - 2017 23rd IEEE International Conference on Automation and Computing: Addressing Global Challenges through Automation and Computing*, Institute of Electrical and Electronics Engineers Inc., Oct. 2017, ISBN: 9780701702618. DOI: 10.23919/IConAC.2017.8082091.

[56] J. Tsai, T. Y. Chang, Y. H. Fang and E. S. Chang, 'A Real-Time Traffic Flow Prediction System for National Freeways Based on the Spark Streaming Technique,' in *2018 IEEE International Conference on Consumer Electronics-Taiwan, ICCE-TW 2018*, Institute of Electrical and Electronics Engineers Inc., Aug. 2018, ISBN: 9781538663011. DOI: 10.1109/ICCE-China.2018.8448998.

[57] C. F. Wu, T. C. Hsu, H. Yang and Y. C. Chung, 'File placement mechanisms for improving write throughputs of cloud storage services based on Ceph and HDFS,' in *Proceedings of the 2017 IEEE International Conference on Applied System Innovation: Applied System Innovation for Modern Technology, ICASI 2017*, Institute of Electrical and Electronics Engineers Inc., Jul. 2017, pp. 1725–1728, ISBN: 9781509048977. DOI: 10.1109/ICASI.2017.7988272.

[58] A. Kirby, B. Henson, J. Thomas, M. Armstrong and M. Galloway, 'Storage and file structure of a bioinformatics cloud architecture,' in *Proceedings - 2019 3rd IEEE International Conference on Cloud and Fog Computing Technologies and Applications, Cloud Summit 2019*, Institute of Electrical and Electronics Engineers Inc.,

Aug. 2019, pp. 110–115, ISBN: 9781728131016. DOI: 10.1109/CloudSummit47114.2019.00024.

[59] C. T. Yang, W. H. Lien, Y. C. Shen and F. Y. Leu, 'Implementation of a Software-Defined Storage Service with Heterogeneous Storage Technologies,' in *Proceedings - IEEE 29th International Conference on Advanced Information Networking and Applications Workshops, WAINA 2015*, Institute of Electrical and Electronics Engineers Inc., Apr. 2015, pp. 102–107, ISBN: 9781479917747. DOI: 10.1109/WAINA.2015.50.

[60] M. Tanaka, O. Tatebe and H. Kawashima, 'Applying Pwrake Workflow System and Gfarm File System to Telescope Data Processing,' in *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, vol. 2018-Septe, Institute of Electrical and Electronics Engineers Inc., Oct. 2018, pp. 124–133, ISBN: 9781538683194. DOI: 10.1109/CLUSTER.2018.00024.

[61] S. Paul, N. Das and B. B. Sarkar, 'Big data infrastructure: Storage considerations,' in *2016 International Conference on Computing, Analytics and Security Trends (CAST)*, IEEE, Dec. 2016, pp. 617–621, ISBN: 978-1-5090-1338-8. DOI: 10.1109/CAST.2016.7915041.

[62] S. Zuozhi, M. Yunlang and Y. Yue, 'Research on Condition Monitoring of Intelligent Substation Equipment Based on Hadoop and MapReduce,' in *Proceedings - 10th International Conference on Intelligent Computation Technology and Automation, ICICTA 2017*, vol. 2017-Octob, Institute of Electrical and Electronics Engineers Inc., Oct. 2017, pp. 402–405, ISBN: 9781538612309. DOI: 10.1109/ICICTA.2017.96.

[63] Y. Benlachmi and M. L. Hasnaoui, 'Big data and spark: Comparison with hadoop,' in *Proceedings of the World Conference on Smart Trends in Systems, Security and Sustainability, WS4 2020*, Institute of Electrical and Electronics Engineers Inc., Jul. 2020, pp. 811–817, ISBN: 9781728168234. DOI: 10.1109/WorldS450073.2020.9210353.

[64] A. Verma, A. H. Mansuri and N. Jain, 'Big data management processing with Hadoop MapReduce and spark technology: A comparison,' in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, IEEE, Mar. 2016, pp. 1–4, ISBN: 978-1-5090-0669-4. DOI: 10.1109/CDAN.2016.7570891.

[65] S. Mishra and C. Hota, 'A REST Framework on IoT Streams using Apache Spark for Smart Cities,' in *2019 IEEE 16th India Council International Conference (INDICON)*, IEEE, Dec. 2019, pp. 1–4, ISBN: 978-1-7281-2327-1. DOI: 10.1109/INDICON47234.2019.9029012.

[66] A. Saraswathi, A. Mummoorthy, A. Raman G.R. and K. Porkodi, 'Real-Time Traffic Monitoring System Using Spark,' in *2019 International Conference on Emerging Trends in Science and Engineering (ICESE)*, IEEE, Sep. 2019, pp. 1–6, ISBN: 978-1-7281-1871-0. DOI: 10.1109/ICESE46178.2019.9194613.

[67] Y. Drohobytskiy, V. Brevus and Y. Skorenkyy, 'Spark structured streaming: Customizing kafka stream processing,' in *Proceedings of the 2020 IEEE 3rd International Conference on Data Stream Mining and Processing, DSMP 2020*, Institute of Electrical and Electronics Engineers Inc., Aug. 2020, pp. 296–299, ISBN: 9781728132143. DOI: 10.1109/DSMP47368.2020.9204304.

[68] Z. Chen, N. Chen and J. Gong, 'Design and implementation of the real-time gis data model and sensor web service platform for environmental big data management with the apache storm,' in *2015 Fourth International Conference on Agro-Geoinformatics (Agro-geoinformatics)*, 2015, pp. 32–35. DOI: 10.1109/Agro-Geoinformatics.2015.7248139.

[69] J. Karimov, T. Rabl, A. Katsifodimos, R. Samarev, H. Heiskanen and V. Markl, 'Benchmarking distributed stream data processing systems,' in *Proceedings - IEEE 34th International Conference on Data Engineering, ICDE 2018*, Institute of Electrical and Electronics Engineers Inc., Feb. 2018, pp. 1519–1530, ISBN: 9781538655207. DOI: 10.1109/ICDE.2018.00169. eprint: 1802.08496.

[70] Z. Karakaya, A. Yazici and M. Alayyoub, 'A Comparison of Stream Processing Frameworks,' in *2017 International Conference on Computer and Applications (ICCA)*, IEEE, Sep. 2017, pp. 1–12, ISBN: 978-1-5386-2752-5. DOI: 10.1109/COMAPP.2017.8079733.

[71] J. Dean and S. Ghemawat, 'Mapreduce: Simplified data processing on large clusters,' in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, 2004, pp. 137–150.

[72] M. Manwal and A. Gupta, 'Big data and hadoop-A technological survey,' in *2017 International Conference on Emerging Trends in Computing and Communication Technologies, ICETCCT 2017*, vol. 2018-Janua, Institute of Electrical and Electronics Engineers Inc., Feb. 2018, pp. 1–6, ISBN: 9781538611470. DOI: 10.1109/ICETCCT.2017.8280345.

[73] T. Sirisakdiwan and N. Nupairoj, 'Spark Framework for Real-Time Analytic of Multiple Heterogeneous Data Streams,' in *2019 2nd International Conference on Communication Engineering and Technology (ICCET)*, IEEE, Apr. 2019, pp. 1–5, ISBN: 978-1-7281-1439-2. DOI: 10.1109/ICCET.2019.8726886.

[74] D. Jayanthi and G. Sumathi, 'Weather data analysis using spark — An in-memory computing framework,' in *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*, vol. 2017-Janua, IEEE, Apr. 2017, pp. 1–5, ISBN: 978-1-5090-5682-8. DOI: 10.1109/IPACT.2017.8245142.

[75] F. Chen, J. Liu and Y. Zhu, 'A Real-Time Scheduling Strategy Based on Processing Framework of Hadoop,' in *2017 IEEE International Congress on Big Data (BigData Congress)*, IEEE, Jun. 2017, pp. 321–328, ISBN: 978-1-5386-1996-4. DOI: 10.1109/BigDataCongress.2017.48.

[76] L. F. Sikos, 'Packet analysis for network forensics: A comprehensive survey,' *Forensic Science International: Digital Investigation*, vol. 32, p. 200 892, Mar. 2020, ISSN: 26662817. DOI: 10.1016/j.fsidi.2019.200892.

[77] H. Shahzad, A. R. Sattar and J. Skandaraniyam, 'DGA Domain Detection using Deep Learning,' in *2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP)*, IEEE, Jan. 2021, pp. 139–143, ISBN: 978-1-7281-8621-4. DOI: 10.1109/CSP51677.2021.9357591.

[78] A. Soleymani and F. Arabgol, 'A novel approach for detecting dga-based botnets in dns queries using machine learning techniques,' *J. Comput. Networks Commun.*, vol. 2021, 4767388:1–4767388:13, 2021.

[79] F. Bisio, S. Saeli, P. Lombardo, D. Bernardi, A. Perotti and D. Massa, 'Real-time behavioral dga detection through machine learning,' in *2017 International Carnahan Conference on Security Technology (ICCST)*, IEEE, 2017, pp. 1–6.

[80] 'A Study of Apache Kafka in Big Data Stream Processing,' in *2018 International Conference on Information , Communication, Engineering and Technology (ICICET)*, IEEE, Aug. 2018, pp. 1–3, ISBN: 978-1-5386-5510-8. DOI: 10.1109/ICICET.2018.8533771. [Online]. Available: https://ieeexplore.ieee.org/document/8533771/.

[81] B. Yu, J. Pan, J. Hu, A. Nascimento and M. De Cock, 'Character Level based Detection of DGA Domain Names,' in *2018 International Joint Conference on Neural Networks (IJCNN)*, vol. 2018-July, IEEE, Jul. 2018, pp. 1–8, ISBN: 978-1-5090-6014-6. DOI: 10.1109/IJCNN.2018.8489147. [Online]. Available: https://ieeexplore.ieee.org/document/8489147/.

Appendix A

Original dissertation proposal

Curso de Mestrado em Informática

Ano letivo de 2020/2021

Sistema de análise de redes de computadores para a identificação de atividade maliciosa

Aluno: Rafael Cardoso de Oliveira

Orientador: Tiago Miguel Ferreira Guimarães Pedrosa

Coorientador: Rui Pedro Sanches de Castro Lopes

1 Objetivo

Esta dissertação visa criar uma solução para capturar, analisar e processar tráfego de rede, com o objetivo de detetar anomalias, como possíveis ataques cibernéticos e/ou a presença de programas maliciosos que prejudicam o bom funcionamento da rede. Será feita uma análise bibliográfica com o objetivo de enquadrar tecnologicamente e cientificamente o trabalho a desenvolver e para servir de base para o desenho de uma arquitetura fiável e escalável para a captura, armazenamento e análise do tráfego.

A analise do tráfego será feita desenvolvendo soluções que, através do comportamento da rede, permita evidenciar situações de potencial atividade maliciosa na rede.

2 Detalhes

Hoje em dia, a maioria das empresas e organizações estão ligadas à internet e, muitas delas, possuem um grande número de equipamentos interligados entre si formando uma ou várias redes. A complexidade e dimensão das redes bem como as técnicas de ataque cada vez mais evoluídas, fazem com que seja difícil verificar se existe atividade maliciosa dentro da rede de uma organização. Muitas delas sofrem ataques cibernéticos sem que se apercebam. Existe igualmente um número bastante elevado de pequenas empresas que encerram permanentemente pouco após um ataque cibernético. Sendo assim, é fundamental, tanto para as pequenas como médias e grandes empresas ou organizações, um sistema que seja capaz de automaticamente reportar para os administradores as atividades maliciosas detetadas.

3 Metodologia de trabalho

Inicialmente será efetuada uma análise bibliográfica, levantando o que atualmente a comunidade está a fazer para resolver certos problemas relativamente à captura, armazenamento e análise de tráfego de rede. De seguida, realizar o levantamento do estado da arte na perspetiva das tecnologias e arquiteturas que se pretende utilizar.

A segunda fase será utilizar as informações levantadas na fase anterior para definir uma arquitetura de

Appendix B

OL2A published paper

A scalable, real-time packet capturing solution

Rafael Oliveira[1[0000−0003−4997−4757]], João P. Almeida[1[0000−0002−1286−2527]], Isabel Praça[2[0000−0002−2519−9859]], Rui Pedro Lopes[1[0000−0002−9170−5078]], and Tiago Pedrosa[1[0000−0003−4873−2705]]

[1] Research Centre in Digitalization and Intelligent Robotics (CeDRI), Instituto Politécnico de Bragança, Portugal {rafael.cardoso,jpa,rlopes,pedrosa}@ipb.pt
[2] ISEP/GECAD Research Centre: Porto, Portugal
icp@isep.ipp.pt

Abstract. The evolution of technology and the increasing connectivity between devices lead to an increased risk of cyberattacks. Good protection systems, such as Intrusion Detection System (IDS) and Intrusion Prevention System (IPS), are essential in trying to prevent, detect and counter most of the attacks. However, the increasing creativity and type of attacks raise the need for more resources and processing power for the protection systems which, in turn, requires horizontal scalability to keep up with the massive companies' network infrastructure and with the complexity of attacks. Technologies like machine learning, show promising results and can be of added value in the detection and prevention of attacks in real-time. But good algorithms and tools are not enough. They require reliable and solid datasets to be able to effectively train the protection systems. The development of a good dataset requires horizontal-scalable, robust, modular and fault-tolerance systems, so that the analyses may be done also in real-time. This paper describes an architecture for horizontal-scaling capture architecture, able to collect packets from multiple sources and prepared for real-time analysis. It depends on multiple modular nodes with specific roles to support different algorithms and tools.

Keywords: packet capture · packet storage · distributed system · machine learning.

1 Introduction

Each year the number of cyberattacks on both companies and individuals rise exponentially, with only a few with the ability to defend themselves and prevent the attacks [18]. To tackle this problem and to foster cyberattacks prevention, it is necessary to use protection tools such as IDS and IPS. However, the number and complexity of attacks pose a challenge to these tools, easily exhausting their storage and processing capacity. Moreover, the sheer amount of data and diversity of vectors of attacks requires systems and tools that are able to scale horizontally, to react in real-time. For that, traffic capturing is fundamental even though being disregarded in many research works [1, 3, 6, 12, 17].

The concern with data capture, transportation and storage are as important as data analysis for real-time analysis. This requires horizontal scalability, in a robust and modular architecture. Moreover, the system must be designed in a way that adding more machines should be an easy task. This paper presents an architecture for a reliable and horizontally scalable packet capture system.

In section 2 some facts about the rise of the internet and cyberattacks are described. Section 3 follows with the concepts and challenges in the packet capture process, how to achieve real-time analysis and some tools to perform the capture. Section 4 presents the different application scenarios and the functional and security requirements that a system should have. Section 5 describes the design of the proposed system, how it works and what technologies it uses.

2 Network and cyberattacks

Nowadays, more and more devices are connected to the internet. It is foreseen that, by the end of 2025, a total of around 8 billion people and 41.2 billion devices are connected to the internet, with 10.3 billion non-IoT devices (laptops, desktops, smartphones, etc.) [10, 20].

With this huge number of devices connected to the Internet, the companies that provide online services (social media, banking, retail, cloud, etc), will also need to expand to be able to keep up with the increasing demand. This growth will consist in an expansion of the companies' Information Technology (IT) infrastructures, adding more servers and routing devices. This increase will also result in a higher probability of suffering a cyberattack, with, eventually, the whole IT infrastructure being compromised. The more devices that are connected to the company infrastructure, the more the risk of suffering a cyberattack.

Cyberattacks are the main problem of the digital world and, according to Lysenko et al. [13], they have generated financial damage of around 1.5 trillion U.S. dollars in 2019. Small companies are the most fragile since around 60% of them close within six months of an incident [7]. But this does not mean that medium to large companies are safe from serious problems, since, for example, leaked intellectual property or stolen user data can have a severe negative impact on any company. CD Projekt, in Feb of 2021, got breached and all of the data stolen (accounting, administration, HR) including the source code of multiple projects (that were sold later by the hackers) not to mention the hours that the employees were unable to work, costing the company even more money [8]. Yahoo, in 2016, announced that in 2013/2014 they suffered a security breach compromising 3 billion user accounts [19], including real names, email addresses, dates of birth and telephone numbers [2], at the time Yahoo was being purchased by another company and after this announcement they lower the offer by 350 million U.S. dollars.

The number of cyberattacks is growing both in number and complexity. There are always new ways of breaching and compromising the networks and with this complexity, traditional IDS and IPS are no longer effective due to the lack of successful detection and prevention of attacks and the ability to operate with complex IT infrastructure. Part of the issue is due to the necessity to capture an increasing number of packets, and to forward them and process them in near real-time.

Efficient data collection systems are, thus, necessary. They must be able to capture data (explanation in this section) then, if the data is not stored in the same machine, they must be transported and, finally, stored securely.

3 Packet capture

Cyberattacks perpetrators usually make efforts to cover their tracks during an attack. Security researchers can find new ways to prevent cyberattacks the same way attackers can adopt anti-forensic techniques trying to remain undetected and without leaving traces.

Log files can be used to detect some attacks, such as massive unauthorized accesses or failed logins. However, they are not enough in most situations, since it is not possible to detect all kind of attacks and there is also the possibility of those getting modified or erased, eluding the security team scrutiny.

There is only one thing that attackers (or anyone else) can never change or purge and that is network traffic. As they can never be changed or removed, it is the best candidate to perform a full-depth analysis of the network, to try to identify who the attackers are, when the attack took place, for how long, with what tools, and what was transferred. Nevertheless, as the network traffic is volatile information, it only exists while transmitted, raising the need to capture and store it in real-time.

As expected, the amount of network traffic is considerable, easily reaching terabyte worth of space in a matter of seconds, making its collection and storage a very expensive operation. Depending on what the security needs to know about the network, it must select which type of data must be stored, from a byte in a header to the full-packet capture.

3.1 Data transportation

In the impossibility of processing locally the data resulting from the packet capture process, it must be transported to a central storage or processing host. Some systems capture and store the packets locally and only after they are transferred to another location for further analyses (usually in PCAP format). However, this approach is not ideal, due to the introduced delay between capturing and analysis. In fact, it does not allow real-time assessment, delaying the discovery of an attack that already happened.

An approach to achieve real-time analyses and also prevent an attack or minimize its damage is to capture the packets and send them immediately to another machine. This way, while a machine is capturing packets and sending them, another one is processing and analysing them.

Nevertheless, it is important that data transportation do not flood the network and if the receiver machine is overloaded, then, the captured machine must have a way to buffer the packets locally.

3.2 Distributed data streaming

To achieve full horizontal scalability, it is fundamental that the destination is not a single host, since it would compromise the possibility to increase the processing power. So, data must be sent to a group of machines that works collectively, in a synchronized way, distributing tasks and workloads.

S. Mousavi et al. [14] used, on their system, Apache Kafka or Redis as a queuing system, to transport data from multiple sources and make them available to multiple destinations. Apache Kafka is a distributed event streaming, capable of synchronizing multiple consumers and producers. Apache Kafka data are modelled as logs and, since logs are events, they are impossible to change or erase because they already happened, making the Apache Kafka reliable and safe from modifications while the data is in the distributed queuing system.

J. Evermann et al. [5] used Amazon Kinesis, another example of a distributed event stream, to process data on the fly, stating that they could process “tens of millions of events per minute”.

3.3 Data collection

Depending on the architecture packet storage may or may not be necessary. With a distributed data streaming system, the data consuming hosts (analyzing modules, algorithms for DNS detection, classification, estimation, prevision, etc.) may consume data directly from the queues in real-time. The queuing system, normally, will persist the data for a predefined period of time and delete the oldest if it runs out of space.

As said before, data collection systems are systems that capture, transmit and store network packets. This kind of systems allows the security team to detect network attacks by searching for abnormal patterns.

P. Emmerich et al. [4] implemented a custom local queuing data structure queue of queues. Their system can capture and store packets in 100Gbit/s networks (120Gbit/s was reached in their experiments). The problem with their solution is that it is not a distributed solution, which can be a bottleneck since they capture and store packets in a local file.

P. Roquero et al. [16] present a scalable data collection system, capturing packets in multiple points of a network and sending them to multiple receivers for analysis. Data capture is performed by a swarm of software probes, that have to be installed in all the network computers. This requires access to each computer and individual installation and configuration, and it would be difficult to implement due to the need for authorization to install the software in critical servers, and the impossibility to install it in IoT devices, embedded systems, and others.

Data capture rely on specialized tools, that can be both software or hardware-based. Software-based packet capture tools consist of several subsystems. The packets flow starts at the Network Card Interface (NIC) and ends up in the userspace subsystem. If any problem occurs in any of the subsystems, packet loss will most likely occur [9]. Some examples are nProbe, ntopng, netsniff-ng, TCPDump and Scapy.

Hardware-based tools are usually more expensive than software-based tools. Data Acquisition and Generation (DAG) cards are devices to capture network packets with filters at the hardware level. In routing devices that have port mirroring activated, they send a copy of every packet that crosses that routing device to a specific physical port. Port mirroring can be enabled without modifying the network infrastructure.

4 Application scenarios

In addition to security auditing and processing, data collection is also important for many other applications. Managing the network is one of them, analysing the retransmission rate of the packets, the loss of connectivity, or network failures detection. It is also possible to monitor the networks' Quality of Service (QoS) [16].

Many kinds of malware perform malicious behaviours like, collecting information, compromising the systems, etc, throughout the network. Therefore, it is essential to also find existing malware in the network and not to just focus on what may come from the outside. S. Pudukotai et al. [15] perform malware analysis using machine learning with an accuracy of 92.21%.

Network data can even be used to compute an estimation of the occupancy of a room. E. Longo et al. [11] made a system with cheap Wi-Fi sniffers to capture Wi-Fi network packets. Then, with only Wi-Fi frames, they can estimate the occupancy of a given room.

H. Lin et al. [9] presents a set o functional and security requirements that a data collection should meet, among which we highlight the following functional requirements: it must be flexible and scalable; it must be capable of dynamically knowing which data to filter and capture; it must be automatic in terms of adaptability; it must be able to store collected data.

Concerning the last functional requirement, we defend that storage of the packets, on a real-time analysis system, may be just for a couple of seconds, while the system has time to perform the needed operations. After that, it might not be necessary to keep them stored.

Taking into account the security requirements [9] we highlight the following: it must be able to prevent data loss and ensure data integrity during the capture and transmission; it must be able to verify the integrity and authenticity of the collected data; it must protect the data against unauthorized users.

5 Proposed system

It is important to design a system that respects the functional and security requirements defined above. This system was designed with the idea of fully horizontal scalability with an easy way to add more physical resources. If more packet capture devices are necessary, it should be possible just to plug in new ones. If more analysis services or algorithms are necessary, it should be easy to add them, and the same happens for the data transport (Fig. 1).

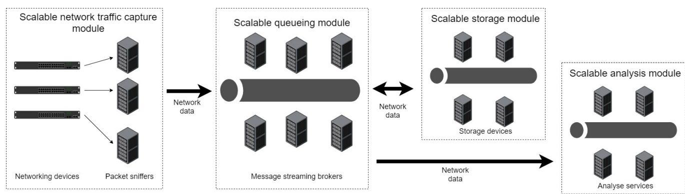

Fig. 1. Components diagram

The number of capture devices depends on the situation. Nevertheless, the strategy is to place a capture device on switch devices that have endpoints connected to it. This way, in the case of Network Address Translation (NAT), the system does not lose any information and knows exactly where the packets came from.

The port mirror approach will be used to make a copy of every packet that crosses that routing device, in a way it's not necessary to modify the current structure of the network. After the capture devices are in place, the system uses the software tool TCPDump to capture the network packets (it captures all packets, not only TCP). The newest versions already use the zero-copy mechanism, which is a good candidate to perform the capture of the packets.

While packets are being captured, they will not be saved locally on that machine, instead, they'll be placed on a local Apache Kafka buffer.

After the system possesses a good continuous dataset of information available in the Apache Kafka ecosystem, any other system (with the right authorization) may consume them, process them and produce results. With that data we can feed any kind of service: it can be sent to dashboard services to monitor the network in real-time, or to a storage system, or to an IDS that uses a set of machine learning algorithms in order to classify the traffic and detect attacks. Again, a solid dataset, growing in real-time, is crucial to develop decision tools and intelligent services to support the security teams.

6 Conclusion

In this work, an approach for network data capturing is presented. The conceptualization is based on relevant scientific literature, assessing the knowledge that emerges from them, and that allows to frame the system in contemporary IT infrastructures and requirements. With this in mind, a flexible, scalable and practical architecture is suggested, keeping a low impact on the network.

We are currently testing the system performance, concerning the network packet capture and streaming to the Apache Kafka cluster.

The next steps include instantiating the architecture in a test scenario, composed of IoT devices, regular workstations, laptops and different servers. This will allow the assessment of the scalability and flexibility of the suggested architecture.

Acknowledgment

This work was partially supported by the Norte Portugal Regional Operational Programme(NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, through the European Regional Development Fund (ERDF), within project "CybersSeCIP" (NORTE-01-0145-FEDER-000044).

References

1. Cordero, C.G., Hauke, S., Muhlhauser, M., Fischer, M.: Analyzing flow-based anomaly intrusion detection using Replicator Neural Networks. In: 2016 14th Annual Conference on Privacy, Security and Trust, PST 2016. pp. 317–324. Institute of Electrical and Electronics Engineers Inc. (2016). https://doi.org/10.1109/PST.2016.7906980
2. Dan Swinhoe: The 15 biggest data breaches of the 21st century — CSO Online (2021), https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html
3. Do, E.H., Gadepally, V.N.: Classifying Anomalies for Network Security. In: ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings. vol. 2020-May, pp. 2907–2911. Institute of Electrical and Electronics Engineers Inc. (may 2020). https://doi.org/10.1109/ICASSP40776.2020.9053419
4. Emmerich, P., Pudelko, M., Gallenmüller, S., Carle, G.: FlowScope: Efficient packet capture and storage in 100 Gbit/s networks. In: 2017 IFIP Networking Conference, IFIP Networking 2017 and Workshops. vol. 2018-Janua, pp. 1–9. Institute of Electrical and Electronics Engineers Inc. (jul 2017). https://doi.org/10.23919/IFIPNetworking.2017.8264852
5. Evermann, J., Rehse, J.R., Fettke, P.: Process discovery from event stream data in the cloud - A scalable, distributed implementation of the flexible heuristics miner on the amazon kinesis cloud infrastructure. In: Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom. vol. 0, pp. 645–652. IEEE Computer Society (jul 2016). https://doi.org/10.1109/CloudCom.2016.0111
6. Guo, Y.T., Gao, Y., Wang, Y., Qin, M.Y., Pu, Y.J., Wang, Z., Liu, D.D., Chen, X.J., Gao, T.F., Lv, T.T., Fu, Z.C.: DPI & DFI: A Malicious Behavior Detection Method Combining Deep Packet Inspection and Deep Flow Inspection. In: Procedia Engineering. vol. 174, pp. 1309–1314. Elsevier Ltd (jan 2017). https://doi.org/10.1016/j.proeng.2017.01.276
7. Johnson, R.: 60 Percent of Small Companies Close Within 6 Months of Being Hacked (2019), https://cybersecurityventures.com/60-percent-of-small-companies-close-within-6-months-of-being-hacked/

8. Kent, E.: CD Projekt hit by ”targeted cyber attack” (2021), https://www.eurogamer.net/articles/2021-02-09-cd-projekt-hit-by-targeted-cyber-attack
9. Lin, H., Yan, Z., Chen, Y., Zhang, L.: A Survey on Network Security-Related Data Collection Technologies. IEEE Access **6**, 18345–18365 (mar 2018). https://doi.org/10.1109/ACCESS.2018.2817921
10. Lionel Sujay Vailsher: Global IoT and non-IoT connections 2010-2025 (2021), https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/
11. Longo, E., Redondi, A.E., Cesana, M.: Accurate occupancy estimation with WiFi and bluetooth/BLE packet capture. Computer Networks **163**, 106876 (nov 2019). https://doi.org/10.1016/j.comnet.2019.106876
12. Lotfollahi, M., Jafari Siavoshani, M., Shirali Hossein Zade, R., Saberian, M.: Deep packet: a novel approach for encrypted traffic classification using deep learning. Soft Computing **24**(3), 1999–2012 (feb 2020). https://doi.org/10.1007/s00500-019-04030-2, https://doi.org/10.1007/s00500-019-04030-2
13. Lysenko, S., Bobrovnikova, K., Shchuka, R., Savenko, O.: A cyberattacks detection technique based on evolutionary algorithms. In: 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT). pp. 127–132 (2020). https://doi.org/10.1109/DESSERT50317.2020.9125016
14. Mousavi, S.H., Khansari, M., Rahmani, R.: A fully scalable big data framework for Botnet detection based on network traffic analysis. Information Sciences **512**, 629–640 (feb 2020). https://doi.org/10.1016/j.ins.2019.10.018
15. Pudukotai Dinakarrao, S.M., Sayadi, H., Makrani, H.M., Nowzari, C., Rafatirad, S., Homayoun, H.: Lightweight Node-level Malware Detection and Network-level Malware Confinement in IoT Networks. In: Proceedings of the 2019 Design, Automation and Test in Europe Conference and Exhibition, DATE 2019. pp. 776–781. Institute of Electrical and Electronics Engineers Inc. (may 2019). https://doi.org/10.23919/DATE.2019.8715057
16. Roquero, P., Magaña, E., Leira, R., Aracil, J.: Performance evaluation of client-based traffic sniffing for very large populations. Computer Networks **166**, 106985 (jan 2020). https://doi.org/10.1016/j.comnet.2019.106985
17. Saini, P.S., Behal, S., Bhatia, S.: Detection of DDoS attacks using machine learning algorithms. In: Proceedings of the 7th International Conference on Computing for Sustainable Global Development, INDIACom 2020. pp. 16–21. Institute of Electrical and Electronics Engineers Inc. (mar 2020). https://doi.org/10.23919/INDIACom49435.2020.9083716
18. Sobers, R.: Data Breach Response Times: Trends and Tips (2020), https://www.varonis.com/blog/data-breach-response-times/
19. Stempel Jonathan, Finkle Jim: Yahoo says all three billion accounts hacked in 2013 data theft — Reuters (2017), https://www.reuters.com/article/us-yahoo-cyber/yahoo-says-all-three-billion-accounts-hacked-in-2013-data-theft-idUSKCN1C82O1
20. Worldometer: World Population Projections - Worldometer (2021), https://www.worldometers.info/world-population/world-population-projections/

Appendix C

Switch configurations

Listing C.1: Cisco switch configuration

```
ip dhcp pool 1
 network 10.0.0.0 255.255.255.0
!
ip dhcp excluded-address 10.0.0.1 10.0.0.9
!
vlan 5
 name mirroring
!
vlan 10
 name management
!
interface GigabitEthernet1/0/20
 switchport mode access
!
interface GigabitEthernet1/0/21
 switchport mode access
!
interface GigabitEthernet1/0/22
 switchport access vlan 5
 switchport mode access
!
interface GigabitEthernet1/0/24
 switchport access vlan 10
 switchport mode access
!
interface Vlan1
```

```
 ip address 10.0.0.1 255.255.255.0
!
interface Vlan10
 ip address 10.1.1.1 255.255.255.0
!
monitor session 1 source vlan 1
monitor session 1 destination interface Gi1/0/22 encapsulation replicate
```

Appendix D

Services deployment

Listing D.1: Zookeeper installation script

```
#!/bin/bash
apt update
apt install default-jdk openjdk-8-jdk -y

cat ../hosts >> /etc/hosts

cd /opt
wget https://mirrors.up.pt/pub/apache/zookeeper/
    zookeeper-3.7.0/apache-zookeeper-3.7.0-bin.tar.gz

tar -xzf apache-zookeeper-3.7.0-bin.tar.gz
rm apache-zookeeper-3.7.0-bin.tar.gz
mv apache-zookeeper-3.7.0-bin apache-zookeeper
chmod 777 apache-zookeeper -R

cd - > /dev/null
cp zoo.cfg /opt/apache-zookeeper/conf/.
cp zookeeper.service /etc/systemd/system/.
mkdir -p /data/zookeeper
chmod 777 -R /data

mkdir -p /ssl
chmod 777 /ssl -R
cp ../ssl/* /ssl/.

systemctl daemon-reload
```

```
systemctl enable zookeeper
systemctl start zookeeper

echo 'Done.'
```

Listing D.2: Zookeeper configuration file

```
tickTime = 2000
dataDir = /data/zookeeper
initLimit = 5
syncLimit = 2
serverCnxnFactory=org.apache.zookeeper.server.NettyServerCnxnFactory
admin.enableServer=false
secureClientPort=2281
ssl.clientAuth=need
ssl.keyStore.location=/ssl/zookeeper.keystore.jks
ssl.keyStore.password=estigestig
ssl.trustStore.location=/ssl/kafka.truststore.jks
ssl.trustStore.password=estigestig
```

Listing D.3: Zookeeper service manager configuration file

```
[Unit]
Description=Zookeeper Daemon
Documentation=http://zookeeper.apache.org
Requires=network.target
After=network.target

[Service]
Type=forking
WorkingDirectory=/opt/apache-zookeeper
ExecStart=/opt/apache-zookeeper/bin/zkServer.sh start /opt/apache-zookeeper/conf/zoo.cfg
ExecStop= /opt/apache-zookeeper/bin/zkServer.sh stop /opt/apache-zookeeper/conf/zoo.cfg
ExecReload=/opt/apache-zookeeper/bin/zkServer.sh restart /opt/apache-zookeeper/conf/zoo.cfg
TimeoutSec=30
Restart=on-failure

[Install]
WantedBy=default.target
```

Listing D.4: Kafka installation script

```
#!/bin/bash

if [ $# -eq 0 ]
  then
    printf "Missing broker id.\nUsage: $0 <id>\n"
    exit
fi

node_id=$1

apt update
apt install default-jdk openjdk-8-jdk -y

cat ../hosts >> /etc/hosts

cd /opt
wget https://dlcdn.apache.org/kafka/2.8.0/kafka_2.13-2.8.0.tgz
tar -xzf kafka_2.13-2.8.0.tgz
rm kafka_2.13-2.8.0.tgz
mv kafka_2.13-2.8.0 apache-kafka
chmod 777 apache-kafka -R

cd - > /dev/null
sed -e "s/\${id}/$node_id/" server.properties > /opt/apache-kafka/config/server.properties
cp kafka.service /etc/systemd/system/.
mkdir -p /data/kafka-logs
chmod 777 -R /data

mkdir -p /ssl
chmod 777 /ssl -R
cp ../ssl/* /ssl/.

systemctl daemon-reload
systemctl enable kafka
systemctl start kafka

echo 'Done.'
```

Listing D.5: Kafka configuration file

```
broker.id=${id}
listeners=SSL://:9093
```

```
security.inter.broker.protocol=SSL
advertised.host.name=kafka${id}.msisc.com
advertised.listeners=SSL://kafka${id}.msisc.com:9093

num.network.threads=3
num.io.threads=8
socket.send.buffer.bytes=102400
socket.receive.buffer.bytes=102400
socket.request.max.bytes=104857600
log.dirs=/data/kafka-logs
num.partitions=1

num.recovery.threads.per.data.dir=1
offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1

log.retention.hours=168
log.segment.bytes=1073741824
log.retention.check.interval.ms=300000

zookeeper.connect=zookeeper.msisc.com:2281
zookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty
zookeeper.ssl.client.enable=true
ssl.client.auth=required
ssl.keystore.location=/ssl/kafka${id}.keystore.jks
ssl.keystore.password=estigestig
zookeeper.ssl.keystore.location=/ssl/kafka${id}.keystore.jks
zookeeper.ssl.keystore.password=estigestig
ssl.truststore.location=/ssl/kafka.truststore.jks
ssl.truststore.password=estigestig
zookeeper.ssl.truststore.location=/ssl/kafka.truststore.jks
zookeeper.ssl.truststore.password=estigestig
```

Listing D.6: Kafka service manager configuration file

```
[Unit]
Description=Apache Kafka Broker
Documentation=http://kafka.apache.org/documentation.html

[Service]
Type=simple
Environment="JAVA_HOME=/usr/lib/jvm/java-1.11.0-openjdk-amd64"
```

```
ExecStart=/opt/apache-kafka/bin/kafka-server-start.sh /opt/apache-kafka/config/server.properties
ExecStop=/opt/apache-kafka/kafka-server-stop.sh

[Install]
WantedBy=multi-user.target
```

Listing D.7: Hadoop name node installation script

```
#!/bin/bash

apt update
apt install default-jdk openjdk-8-jdk -y
cat ../../hosts >> /etc/hosts

cd /opt
wget https://dlcdn.apache.org/hadoop/common/hadoop-3.2.2/hadoop-3.2.2.tar.gz
tar -xzf hadoop-3.2.2.tar.gz
rm hadoop-3.2.2.tar.gz
mv hadoop-3.2.2 apache-hadoop
chmod 777 apache-hadoop -R

cd - > /dev/null

cd ..
cp core-site.xml hadoop-env.sh hdfs-site.xml /opt/apache-hadoop/etc/hadoop/
/opt/apache-hadoop/bin/hdfs namenode -format

cd - > /dev/null

chmod +x start-hdfs.sh stop-hdfs.sh
cp start-hdfs.sh stop-hdfs.sh /opt/apache-hadoop/
cp hadoop.service /etc/systemd/system/.

mkdir -p /ssl
chmod 777 /ssl -R
cp ../../ssl/* /ssl/.

systemctl daemon-reload
systemctl enable hadoop
systemctl start hadoop

echo 'Done.'
```

Listing D.8: Hadoop name node service manager configuration file

```
[Unit]
Description=HDFS Daemon
Documentation=http://zookeeper.apache.org
Requires=network.target
After=network.target

[Service]
Type=forking
WorkingDirectory=/opt/apache-hadoop/bin/
ExecStart=/opt/apache-hadoop/start-hdfs.sh
ExecStop=/opt/apache-hadoop/stop-hdfs.sh

TimeoutSec=30
Restart=on-failure

[Install]
WantedBy=default.target
```

Listing D.9: Hadoop data node installation script

```
#!/bin/bash

apt update
apt install default-jdk openjdk-8-jdk -y

cat ../../hosts >> /etc/hosts
cd /opt
wget https://dlcdn.apache.org/hadoop/common/hadoop-3.2.2/hadoop-3.2.2.tar.gz
tar -xzf hadoop-3.2.2.tar.gz
rm hadoop-3.2.2.tar.gz
mv hadoop-3.2.2 apache-hadoop
chmod 777 apache-hadoop -R

cd - > /dev/null

cp yarn-site.xml /opt/apache-hadoop/etc/hadoop/

cd ..
cp core-site.xml hadoop-env.sh hdfs-site.xml /opt/apache-hadoop/etc/hadoop/
```

```
/opt/apache-hadoop/bin/hdfs datanode -format

cd - > /dev/null

cp hadoop.service /etc/systemd/system/.

mkdir -p /ssl
chmod 777 /ssl -R
cp ../../ssl/* /ssl/.

systemctl daemon-reload
systemctl enable hadoop
systemctl start hadoop

echo 'Done.'
```

Listing D.10: Hadoop data node service manager configuration file

```
[Unit]
Description=HDFS Daemon
Documentation=http://zookeeper.apache.org
Requires=network.target
After=network.target

[Service]
Type=forking
WorkingDirectory=/opt/apache-hadoop/bin/
ExecStart=/opt/apache-hadoop/bin/hdfs --daemon start datanode
ExecStop=/opt/apache-hadoop/bin/hdfs --daemon stop datanode

TimeoutSec=30
Restart=on-failure

[Install]
WantedBy=default.target
```

Listing D.11: Hadoop name and data node configuration file (core-site.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
```

```
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://hadoop-master.msisc.com:9000</value>
  </property>
  <property>
    <name>hadoop.ssl.require.client.cert</name>
    <value>false</value>
  </property>
  <property>
    <name>hadoop.ssl.hostname.verifier</name>
    <value>DEFAULT</value>
  </property>
  <property>
    <name>hadoop.ssl.keystores.factory.class</name>
    <value>org.apache.hadoop.security.ssl.FileBasedKeyStoresFactory</value>
  </property>
  <property>
    <name>hadoop.ssl.server.conf</name>
    <value>ssl-server.xml</value>
  </property>
  <property>
    <name>hadoop.ssl.client.conf</name>
    <value>ssl-client.xml</value>
  </property>
</configuration>
```

Listing D.12: Hadoop name and data node node configuration file (hdfs-site.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>dfs.permissions</name>
    <value>false</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>/opt/apache-hadoop/data/nameNode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>/opt/apache-hadoop/data/dataNode</value>
  </property>
```

```
  <property>
    <name>dfs.replication</name>
    <value>2</value>
  </property>
  <property>
    <name>dfs.http.policy</name>
    <value>HTTPS_ONLY</value>
  </property>
  <property>
    <name>dfs.client.https.need-auth</name>
    <value>false</value>
  </property>
  <property>
    <name>dfs.namenode.https-address</name>
    <value>hadoop-master.msisc.com:50470</value>
  </property>
</configuration>
```

Listing D.13: Hadoop name and data node configuration file (mapred-site.xml)

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>mapreduce.jobhistory.http.policy</name>
    <value>HTTPS_ONLY</value>
  </property>
  <property>
    <name>mapreduce.jobhistory.webapp.https.address</name>
    <value>hadoop-master.msisc.com:19889</value>
  </property>
  <property>
    <name>mapreduce.ssl.enabled</name>
    <value>true</value>
  </property>
  <property>
    <name>mapreduce.shuffle.ssl.enabled</name>
    <value>true</value>
  </property>
</configuration>
```

Listing D.14: Hadoop name and data node configuration file (yarn-site.xml)

```xml
<?xml version="1.0"?>
<configuration>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>hadoop-master.msisc.com</value>
  </property>
  <property>
    <name>yarn.http.policy</name>
    <value>HTTPS_ONLY</value>
  </property>
  <property>
    <name>yarn.log.server.url</name>
    <value>https://hadoop-master.msisc.com:19889</value>
  </property>
  <property>
    <name>yarn.resourcemanager.webapp.https.address</name>
    <value>hadoop-master.msisc.com:8089</value>
  </property>
  <property>
    <name>yarn.nodemanager.webapp.https.address</name>
    <value>0.0.0.0:8090</value>
  </property>
</configuration>
```

Listing D.15: Hadoop name node configuration file (ssl-server.xml)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>ssl.server.truststore.location</name>
    <value>/ssl/kafka.truststore.jks</value>
  </property>
  <property>
    <name>ssl.server.truststore.password</name>
    <value>estigestig</value>
  </property>
  <property>
    <name>ssl.server.truststore.type</name>
    <value>jks</value>
  </property>
  <property>
```

```
    <name>ssl.server.truststore.reload.interval</name>
    <value>10000</value>
  </property>
  <property>
    <name>ssl.server.keystore.location</name>
    <value>/ssl/hadoop-master.keystore.jks</value>
  </property>
  <property>
    <name>ssl.server.keystore.password</name>
    <value>estigestig</value>
  </property>
  <property>
    <name>ssl.server.keystore.keypassword</name>
    <value>estigestig</value>
  </property>
  <property>
    <name>ssl.server.keystore.type</name>
    <value>jks</value>
  </property>
  <property>
    <name>ssl.server.exclude.cipher.list</name>
    <value>TLS_ECDHE_RSA_WITH_RC4_128_SHA,SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA,
  SSL_RSA_WITH_DES_CBC_SHA,SSL_DHE_RSA_WITH_DES_CBC_SHA,
  SSL_RSA_EXPORT_WITH_RC4_40_MD5,SSL_RSA_EXPORT_WITH_DES40_CBC_SHA,
  SSL_RSA_WITH_RC4_128_MD5</value>
  </property>
</configuration>
```

Listing D.16: Hadoop data node configuration file (client-server.xml)

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>ssl.client.truststore.location</name>
    <value>/ssl/kafka.truststore.jks</value>
  </property>
  <property>
    <name>ssl.client.truststore.password</name>
    <value>estigestig</value>
  </property>
  <property>
    <name>ssl.client.truststore.type</name>
```

```
    <value>jks</value>
  </property>
  <property>
    <name>ssl.client.truststore.reload.interval</name>
    <value>10000</value>
  </property>
  <property>
    <name>ssl.client.keystore.location</name>
    <value>/ssl/hadoop-master.keystore.jks</value>
  </property>
  <property>
    <name>ssl.client.keystore.password</name>
    <value>estigestig</value>
  </property>
  <property>
    <name>ssl.client.keystore.keypassword</name>
    <value>estigestig</value>
  </property>
  <property>
    <name>ssl.client.keystore.type</name>
    <value>jks</value>
  </property>
</configuration>
```

Listing D.17: Spark-master installation script

```
#!/bin/bash

apt update
apt install default-jdk openjdk-8-jdk -y

cat ../../hosts >> /etc/hosts

cd /opt
wget https://mirrors.up.pt/pub/apache/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz
tar -xzf spark-3.1.2-bin-hadoop3.2.tgz
rm spark-3.1.2-bin-hadoop3.2.tgz
mv spark-3.1.2-bin-hadoop3.2 apache-spark
chmod 777 apache-spark -R

cd - > /dev/null

cp spark-master.service /etc/systemd/system/.
```

```

mkdir -p /ssl
chmod 777 /ssl -R
cp ../../ssl/* /ssl/.

systemctl daemon-reload
systemctl enable spark-master
systemctl start spark-master

echo 'Done.'
```

Listing D.18: Spark-master service manager configuration file

```
[Unit]
Description=Apache Spark
Documentation=http://kafka.apache.org/documentation.html

[Service]
Type=forking
Environment="JAVA_HOME=/usr/lib/jvm/java-1.11.0-openjdk-amd64"
ExecStart=/opt/apache-spark/sbin/start-master.sh --host spark-master.msisc.com
ExecStop=/opt/apache-spark/sbin/stop-master.sh

[Install]
WantedBy=multi-user.target
```

Listing D.19: Spark-worker installation script

```
#!/bin/bash

apt update
apt install default-jdk openjdk-8-jdk -y

cat ../../hosts >> /etc/hosts

cd /opt
wget https://mirrors.up.pt/pub/apache/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz
tar -xzf spark-3.1.2-bin-hadoop3.2.tgz
rm spark-3.1.2-bin-hadoop3.2.tgz
mv spark-3.1.2-bin-hadoop3.2 apache-spark
chmod 777 apache-spark -R
```

```
cd - > /dev/null

cp spark-slave.service /etc/systemd/system/.

mkdir -p /ssl
chmod 777 /ssl -R
cp ../../ssl/* /ssl/.

systemctl daemon-reload
systemctl enable spark-slave
systemctl start spark-slave

echo 'Done.'
```

Listing D.20: Spark-worker service manager configuration file

```
[Unit]
Description=Apache Spark
Documentation=http://kafka.apache.org/documentation.html

[Service]
Type=forking
Environment="JAVA_HOME=/usr/lib/jvm/java-1.11.0-openjdk-amd64"
ExecStart=/opt/apache-spark/sbin/start-worker.sh spark://spark-master.msisc.com:7077
ExecStop=/opt/apache-spark/sbin/stop-master.sh

[Install]
WantedBy=multi-user.target
```

Appendix E

Packet capture application versions source code

Listing E.1: Packet capture version 2 application complete source-code

```
#define _GNU_SOURCE
#ifdef linux

#include <syscall.h>

#endif

#include <time.h>
#include <sys/time.h>
#include <unistd.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <pcap.h>
#include <strings.h>
#include <string.h>
#include <errno.h>
#include <signal.h>
#include <sched.h>
#include <sys/file.h>
#include <sys/mman.h>
#include <sys/resource.h>
#include <fcntl.h>
```

```c
#include <limits.h>
#include <sys/wait.h>
#include "settings.h"
#include <sys/stat.h>

#include <librdkafka/rdkafka.h>

#define gettid() syscall(__NR_gettid)    /* missing in headers? */
#define MAXPKT 16384        /* larger than any jumbogram */
#define SNAP_LEN 65535        /* apparently what tcpdump uses for -s 0  */
#define WRITESIZE 65536        /* usual write chunk size - must be 2^N (what we want: 524288) */
#define PACKET_BUFFER_TIMEOUT 1000 /* set at 1000 works (original is 0)*/
#define GRE_HDRLEN 50        /* Cisco GRE encapsulation header size */
#define READ_PRIO    -15    /* niceness value for Reader thread */
#define WRITE_PRIO    10    /* niceness value for Writer thread */
#define READER_CPU    1    /* assign Reader thread to this CPU */
#define WRITER_CPU    0    /* assign Writer thread to this CPU */
#define POLL_USECS    1000    /* ring full/empty poll interval */
#ifdef RHEL3
# define my_sched_setaffinity(a,b,c) sched_setaffinity(a, c)
#else
# define my_sched_setaffinity(a, b, c) sched_setaffinity(a, b, c)
#endif /* RHEL3 */
#define V_WIDTH        10    /* minimum size of -V ps status field */
#define TEMPLATE "/gulp.XXXXXX"    /* mktemp template for files in -o dir */

#define RMEM_MAX "/proc/sys/net/core/rmem_max"        /* system tuning */
#define RMEM_DEF "/proc/sys/net/core/rmem_default"    /* system tuning */
#define RMEM_SUG 4194304                /* suggested value */
FILE *procf;
int rmem_def = RMEM_SUG, rmem_max = RMEM_SUG;    /* check tuning */

int WriteSize = WRITESIZE;    /* desired size for aligned writes */
int snap_len = SNAP_LEN;    /* requested limit on packet capture size */
int d_snap_len = SNAP_LEN;    /* actual limit on packet capture size */
int poll_usecs = POLL_USECS;    /* ring full/empty poll interval */
int packet_buffer_timeout = PACKET_BUFFER_TIMEOUT;
int just_copy = 0;        /* read from stdin instead of eth# */
int captured = 0;        /* number of packets captured for stats */
int ignored = 0;        /* number of packets !decapsulated for stats */
int maxbuffered = 0;        /* maximum number of bytes ring buffered */
int ringsize = RINGSIZE;    /* ring buffer size */
int gre_hdrlen = 0;        /* decapsulation header length */
char *dev = "eth1";        /* capture interface device name */
```

```c
char *filter_exp = "";        /* decapsulation filter expression */
char *buf;            /* pointer to the big malloc'd ring buffer */
int volatile start, end;    /* index of first, next byte in buf */
int volatile boundary = -2;    /* index in buf to start a new output file */
int push, eof;            /* flags for inter-thread communication */
char *progname;            /* argv[0] for error messages from threads */
int warn_buf_full = 1;        /* unless reading a file, warn if buf fills */
pcap_t *handle = 0;        /* packet capture handle */
struct pcap_stat pcs;        /* packet capture filter stats */
int got_stats = 0;        /* capture stats have been obtained */
char *id = "@(#) Gulp RCS $Revision: 1.58-crox $"; /* automatically maintained */
int check_eth = 1;        /* check that we are capturing from an Ethernet device */
int would_block = 0;        /* for academic interest only */
int check_block = 0;        /* use select to see if writes would block */
int yield_if_blocking = 0;    /* experimental: may help on uniprocessors */
char *ps_stat_ptr = 0;        /* loc to display buf percentage used */
int ps_stat_len = 0;        /* initial length of -V arg */
int xlock = 0;            /* set if exclusive lock requested */
int lockfd;            /* open descriptor to file to lock */
char *odir = 0;            /* requested output directory name */
char wfile[PATH_MAX];        /* output filename */
char *oname = "pcap";        /* requested output file name */
int tflag = 0;            /* append timestamp to the file name */
int filec = 0;            /* output file number */
struct pcap_file_header fh;    /* begins every pcap file */
int split_after = 10;        /* start new output file after # ringbufs */
int split_seconds = 0;        /* start new output file after # seconds */
time_t bdry_time = 0;            /* packet capture output file open time */
int time_split = 0;        /* 1 when time() - bdry_time > split_seconds */
int max_files = 0;        /* upper bound on filec */
int volatile reader_ready = 0;    /* reader thread no longer needs root */
char *zcmd = NULL;              /* processes each savefile using a specified command */
int zflag = 0;

/* logging purposes */
struct tm *date_info;
time_t raw_time;
struct timeval CAPTURE_STARTED;
struct timeval CAPTURE_ENDED;
struct timeval KAFKA_ENDED;
int PACKETS_CAPTURED, PACKETS_DROPPED, PACKETS_RECEIVED_BY_FILTER;


static void child_cleanup(int); /* to avoid zombies, see below */
```

```c
void kafka_produce_message(char *, int);

rd_kafka_t *rk;

/*
 * put data onto the end of global ring buffer "buf"
 */
void append(char *ptr, int len, int bdry) {
    static int just_wrapped = 0;
    static int wrap_cnt = 0;
    int avail, used;
    static int warned = -1;
    used = end - start;
    if (used < 0) used += ringsize;
    if (used > maxbuffered) maxbuffered = used;
    avail = ringsize - used;

    while (len >= avail) {        /* ring buffer is full, wait */
        if (warned < push) {
            warned = push;
            if (warn_buf_full)
                fprintf(stderr, "%s: ring buffer full\n", progname);
        }
        usleep(poll_usecs);
        used = end - start;
        if (used < 0) used += ringsize;
        avail = ringsize - used;
        if (eof) return;
    }
    if (len > 0 && len < avail) {    /* ring buffer space available */
        if (bdry && (split_seconds != 0) && ((time(NULL) - bdry_time) >= split_seconds)) {
            time_split = 1;
        }
        if (end + len <= ringsize) {      /* no wrap to beginning needed */
            memcpy(buf + end, ptr, len);
        } else {                  /* append wraps */
            int c = ringsize - end;
            memcpy(buf + end, ptr, c);
            memcpy(buf, ptr + c, len - c);
        }
        if (end + len >= ringsize) {
            end += len - ringsize;
            just_wrapped = 1;
        } else {
```

```
            end += len;
        }
        if (time_split || (just_wrapped && bdry)) {
            if (just_wrapped) {
                wrap_cnt++;
                just_wrapped = 0;
            }
            if (odir && (wrap_cnt >= split_after || time_split)) {
                while (boundary >= 0) {      /* last split still pending */
                    if (warned < push) {
                        warned = push;
                        if (warn_buf_full)
                            fprintf(stderr, "%s: ring buffer full\n", progname);
                    }
                    usleep(poll_usecs);
                }
                /*
                 * Tell Writer to start a new file.  Boundary is now < 0 so
                 * last split is complete.  Set boundary BEFORE appending file
                 * header; the write can't happen until the data is appended.
                 */
                boundary = end;
                wrap_cnt = 0;
                bdry_time = time(NULL);
                time_split = 0;
                if (!just_copy) append((char *) &fh, sizeof(fh), 0);
            }
        }
    }
}

#ifndef JUSTCOPY

void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet) {
    struct pcap_pkthdr ph = *header;
    if (ph.caplen >= gre_hdrlen) {    /* sanity test */
        ++captured;
        ph.caplen -= gre_hdrlen;
        ph.len -= gre_hdrlen;

        if (sizeof(long) > sizeof(int) && sizeof(int) > sizeof(short)) {
            struct timeval_32 {
                int tv_sec;
                int tv_usec;
```

```
            } tv32;
            tv32.tv_sec = ph.ts.tv_sec;
            tv32.tv_usec = ph.ts.tv_usec;
            append((char *) &tv32, sizeof(tv32), 0);
            append((char *) &ph + sizeof(struct timeval),
                   sizeof(struct pcap_pkthdr) - sizeof(struct timeval), 0);
        } else
            append((char *) &ph, sizeof(struct pcap_pkthdr), 0);
        append((char *) packet + gre_hdrlen, ph.caplen, 1);
    } else
        ++ignored;
}

#endif /* JUSTCOPY */

void cleanup(int signo) {
    eof = 1;
    if (just_copy == 1 || got_stats) return;
#ifndef JUSTCOPY
#ifndef RHEL3
    pcap_breakloop(handle);
#endif
    if (pcap_stats(handle, &pcs) < 0) {
        if (strcmp(dev, "-"))    /* ignore message if input is stdin */
            (void) fprintf(stderr, "pcap_stats: %s\n", pcap_geterr(handle));
    } else got_stats = 1;
#ifdef RHEL3
    pcap_close(handle);
#endif /* RHEL3 */
#endif /* JUSTCOPY */
}

/*
 * This thread reads stdin or the network and appends to the ring buffer
 */
void *Reader(void *arg) {
#ifndef JUSTCOPY
    char errbuf[PCAP_ERRBUF_SIZE];    /* error buffer */
    struct bpf_program fp;        /* compiled filter program */
    bpf_u_int32 mask;            /* subnet mask */
    bpf_u_int32 net;            /* ip */
    int num_packets = -1;        /* number of packets to capture */
#endif
#ifdef CPU_SET
```

```
    int rtid = gettid();        /* reader thread id */
    cpu_set_t csmask;
    CPU_ZERO(&csmask);
    CPU_SET(READER_CPU, &csmask);
    if (my_sched_setaffinity(rtid, sizeof(cpu_set_t), &csmask) != 0) {
        fprintf(stderr, "%s: Reader could not set cpu affinity: %s\n",
                progname, strerror(errno));
    }
    if (setpriority(PRIO_PROCESS, rtid, READ_PRIO) != 0) {
        fprintf(stderr, "%s: Reader could not set scheduling priority: %s\n",
                progname, strerror(errno));
    }
#else
    replace with equivalent code for your OS or delete and run less optimally
#endif

#ifdef USE_SIGNAL
    signal(SIGINT, cleanup);
    signal(SIGPIPE, cleanup);
#else
    struct sigaction sa;
    sa.sa_handler = cleanup;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;            /* allow signal to abort pcap read */

    sigaction(SIGINT, &sa, NULL);
    sigaction(SIGPIPE, &sa, NULL);
#endif /* USE_SIGNAL */

    if (just_copy) {
        static char rbuf[MAXPKT];
        int c;
        reader_ready = 1;
        while (!eof && (c = read(0, rbuf, MAXPKT)) != 0) {
            if (c > 0) append(rbuf, c, 1);
        }
    }
#ifndef JUSTCOPY
    else {

        /*
         * get network number and mask associated with capture device
         * (needed to compile a bpf expression).
         */
```

```
        if (strcmp(dev, "-") && pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
            fprintf(stderr, "%s: Couldn't get netmask for dev %s: %s\n",
                    progname, dev, errbuf);
            net = 0;
            mask = 0;
        }

        /* open capture device */
        if (!strcmp(dev, "-")) {
            handle = pcap_open_offline(dev, errbuf);
#ifndef RHEL3
            int sfd = -2;
            if (handle) sfd = pcap_get_selectable_fd(handle);
            if (sfd >= 0 && lseek(sfd, 0, SEEK_CUR) >= 0) {
                warn_buf_full = 0;        /* input is a file, don't warn */
            }
#endif /* RHEL3 */
        } else
            handle = pcap_open_live(dev, d_snap_len, 1, packet_buffer_timeout, errbuf);
        if (handle == NULL) {
            fprintf(stderr, "%s: Couldn't open device %s: %s\n",
                    progname, dev, errbuf);
            exit(EXIT_FAILURE);
        }

        reader_ready = 1;

        /* make sure we're capturing on an Ethernet device */
        if (check_eth == 1 && pcap_datalink(handle) != DLT_EN10MB) {
            fprintf(stderr, "%s: %s is not an Ethernet\n", progname, dev);
            exit(EXIT_FAILURE);
        }

        /* compile the filter expression */
        if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
            fprintf(stderr, "%s: Couldn't parse filter %s: %s\n",
                    progname, filter_exp, pcap_geterr(handle));
            exit(EXIT_FAILURE);
        }

        /* apply the compiled filter */
        if (pcap_setfilter(handle, &fp) == -1) {
            fprintf(stderr, "%s: Couldn't install filter %s: %s\n",
                    progname, filter_exp, pcap_geterr(handle));
```

```c
            exit(EXIT_FAILURE);
        }

        /*
         * emit pcap file header
         */

#ifndef RHEL3
        char tmpstr[] = "/tmp/gulp_hdr.XXXXXX";
        int tmpfd = mkstemp(tmpstr);
        if (tmpfd >= 0) {
            pcap_dumper_t *dump = pcap_dump_fopen(handle, fdopen(tmpfd, "w"));
            if (dump)
                pcap_dump_close(dump);
            tmpfd = open(tmpstr, O_RDONLY);    /* get pcap to create a header */
            if (tmpfd >= 0)
                read(tmpfd, (char *) &fh, sizeof(fh));
            if (tmpfd >= 0)
                close(tmpfd);
            unlink(tmpstr);
            fh.snaplen = snap_len;        /* snaplen after any decapsulation */
        }
#endif /* RHEL3 */
        if (fh.magic != 0xa1b2c3d4) {    /* if the above failed, do this */
            fprintf(stderr, "%s: using canned pcap header\n", progname);
            fh.magic = 0xa1b2c3d4;
            fh.version_major = 2;
            fh.version_minor = 4;
            fh.thiszone = 0;
            fh.sigfigs = 0;
            fh.snaplen = snap_len;
            fh.linktype = 1;
        }

        append((char *) &fh, sizeof(fh), 0);

        /* register the start of the capture */
        gettimeofday(&CAPTURE_STARTED, NULL);

        fprintf(stderr, "Capturing...\n");

        /* now we can set our callback function */
        pcap_loop(handle, num_packets, got_packet, NULL);

```

```
        gettimeofday(&CAPTURE_ENDED, NULL);

        fprintf(stderr, "\n%d packets captured\n", captured);
        PACKETS_CAPTURED = captured;

        if (ignored > 0) {
            fprintf(stderr, "%d packets ignored (too small to decapsulate)\n",
                    ignored);
        }
        if (got_stats) {
            (void) fprintf(stderr, "%d packets received by filter\n", pcs.ps_recv);
            (void) fprintf(stderr, "%d packets dropped by kernel\n", pcs.ps_drop);
            PACKETS_RECEIVED_BY_FILTER = pcs.ps_recv;
            PACKETS_DROPPED = pcs.ps_drop;

            /*
             * if packets dropped, check/warn if pcap socket buffer is too small
             */
            if (pcs.ps_drop > 0) {
                procf = fopen(RMEM_DEF, "r");
                if (procf) {
                    fscanf(procf, "%d", &rmem_def);
                    fclose(procf);
                }
                procf = fopen(RMEM_MAX, "r");
                if (procf) {
                    fscanf(procf, "%d", &rmem_max);
                    fclose(procf);
                }
                if (rmem_def < RMEM_SUG || rmem_max < RMEM_SUG) {
                    fprintf(stderr, "\nNote %s may drop fewer packets "
                                    "if you increase:\n  %s and\n  %s\nto %d or more\n\n",
                            progname, RMEM_MAX, RMEM_DEF, RMEM_SUG);
                }
            }
        }
        if (check_block) {
            if (would_block)
                fprintf(stderr, "select reports writes would have blocked\n");
            else
                fprintf(stderr, "select reports writes would not have blocked\n");
        }
        /* cleanup */
        pcap_freecode(&fp);
```

```c
#ifndef RHEL3
        pcap_close(handle);
#endif /* RHEL3 */

    }
#endif /* JUSTCOPY */
    fprintf(stderr, "ring buffer use: %.1lf%% of %d MB\n",
            100.0 * (double) maxbuffered / (double) (ringsize), ringsize / 1024 / 1024);

    eof = 1;

    /* logs */

    struct stat st = {0};
    if (stat("logs", &st) == -1) {
        mkdir("logs", 0600);
    }

    char file_name[100];
    sprintf(file_name, "logs/producer_%d-%d-%d:%d_%d_%d.json", 1900 + date_info->tm_year, 1 + date_info->tm_mon,
            date_info->tm_mday, date_info->tm_hour, date_info->tm_min, date_info->tm_sec);

    FILE *json_file = fopen(file_name, "w");
    fprintf(json_file, "{\n\"method\": \"gulp\",\n\"capture_started\": %ld.%ld,\n", CAPTURE_STARTED.tv_sec,
            CAPTURE_STARTED.tv_usec);
    fprintf(json_file, "\"capture_ended\": %ld.%ld,\n", CAPTURE_ENDED.tv_sec, CAPTURE_ENDED.tv_usec);
    fprintf(json_file, "\"kafka_ended\": %ld.%ld,\n", KAFKA_ENDED.tv_sec, KAFKA_ENDED.tv_usec);
    fprintf(json_file, "\"snaplen\": %d,\n", snap_len);
    fprintf(json_file, "\"kafka_chunks_bytes\": %d,\n", WriteSize);
    fprintf(json_file, "\"packets_captured\": %d,\n", PACKETS_CAPTURED);
    fprintf(json_file, "\"packets_dropped\": %d,\n", PACKETS_DROPPED);
    fprintf(json_file, "\"packets_received_by_filter\": %d\n}", PACKETS_RECEIVED_BY_FILTER);
    fclose(json_file);

    fflush(stderr);
    pthread_exit(NULL);
}

/*
 * Post-process capture files after they have been rotated
 * (copied from tcpdump)
 */
static void child_cleanup(int signo) {
```

```
    wait(NULL);
}

void process_savefile(char filename[PATH_MAX]) {
    pid_t pid;

    if (!(zflag && strlen(filename)))
        return;

    pid = fork();
    if (pid == -1) {
        fprintf(stderr, "process_savefile: fork(): %s\n", strerror(errno));
        return;
    } else if (pid > 0) {
        /* parent process */
        return;
    }

    /* set to lowest priority */
#ifdef NZERO
    setpriority(PRIO_PROCESS, 0, NZERO - 1);
#else
    setpriority(PRIO_PROCESS, 0, 19);
#endif
    execlp(zcmd, zcmd, filename, (char *) NULL);
    /* exec*() return only on failure */
    fprintf(stderr, "process_savefile: execlp(%s, %s): %s\n", zcmd, filename, strerror(errno));
    exit(EXIT_FAILURE);
}

/*
 * Redirect standard output into a new capture file in the specified directory.
 *
 * In case Gulp is running setuid root, try to prevent a user from
 * overwriting system files.  This is accomplished by creating output files
 * with random temporary names in a directory to which the user has write
 * access and subsequently renaming them to names unlikely to cause trouble.
 */
int newoutfile(char *dir, int num) {
    char tfile[PATH_MAX];        /* output temp filename */
    char ofile[PATH_MAX];        /* output real filename */
    if (access(dir, W_OK) != 0) {
        if (access(dir, F_OK) != 0) {
            fprintf(stderr, "%s: -o dir does not exist: '%s'\n",
```

```
                    progname, dir);
            return (0);
        }
        fprintf(stderr, "%s: can't create files in '%s'\n", progname, dir);
        return (0);
    }
    snprintf(tfile, sizeof(tfile), "%s%s", dir, TEMPLATE);
    if (tflag) {
//      snprintf(ofile, sizeof(ofile), "%s/%s%lld.%03d", dir, oname, (long long int)time(NULL), num);
        char outstr[200];
        time_t t;
        struct tm *tmp;
        const char *fmt = "%Y%m%d%H%M%S";

        t = time(NULL);
        tmp = gmtime(&t);
        if (tmp == NULL) {
            perror("gmtime error");
            exit(EXIT_FAILURE);
        }

        if (strftime(outstr, sizeof(outstr), fmt, tmp) == 0) {
            fprintf(stderr, "strftime returned 0");
            exit(EXIT_FAILURE);
        }

        snprintf(ofile, sizeof(ofile), "%s/%s_%s.pcap", dir, oname, outstr);
    } else {
        snprintf(ofile, sizeof(ofile), "%s/%s%03d.pcap", dir, oname, num);
    }
    int tmpfd = mkstemp(tfile);
    fchown(tmpfd, getuid(), -1);    /* in case running setuid */
    if (tmpfd >= 0) {
        if (freopen(tfile, "w", stdout) == NULL) {
            fprintf(stderr, "%s: can't create output file: '%s'\n",
                    progname, tfile);
            return (0);
        }
        dup2(tmpfd, fileno(stdout));    /* try to use the initial fd */
        close(tmpfd);
        rename(tfile, ofile);
        if (odir) process_savefile(wfile);
        /* wfile = ofile; */
        snprintf(wfile, sizeof(wfile), "%s", ofile);
```

```
        return (1);
    } else {
        fprintf(stderr, "%s: can't create: '%s'\n", progname, tfile);
        return (0);
    }
    return (0);                 /* some error */
}

/*
 * This thread copies the ring buffer to stdout in WriteSize chunks
 * or every second (or so) whichever happens first.
 */
void *Writer(void *arg) {
    int n;
    int used;
    int writesize;
    int done = 0;
    int pushed = 0;             /* value of "push" at last write  */
#ifdef CPU_SET
    int wtid = gettid();         /* Writer thread id */
    cpu_set_t csmask;
    CPU_ZERO(&csmask);
    CPU_SET(WRITER_CPU, &csmask);
    if (my_sched_setaffinity(wtid, sizeof(cpu_set_t), &csmask) != 0) {
        fprintf(stderr, "%s: Writer could not set cpu affinity: %s\n",
                progname, strerror(errno));
    }
    if (setpriority(PRIO_PROCESS, wtid, WRITE_PRIO) != 0) {
        fprintf(stderr, "%s: Writer could not set scheduling priority: %s\n",
                progname, strerror(errno));
    }
#else
    replace with equivalent code for your OS or delete and run less optimally
#endif /* CPU_SET */

    if (geteuid() != getuid()) {
        while (!reader_ready) usleep(poll_usecs);
        seteuid(getuid());        /* drop setuid privilege */
    }

    if (tflag) {
        if (max_files && max_files != 1000) {
            fprintf(stderr, "%s: -W will be set to 1000 because -t is also set\n", progname);
        }
```

```c
        max_files = 1000;
    }

    if (odir && !newoutfile(odir, filec++)) {
        exit(1);
    }

    while (!done) {
        used = end - start;
        if (used < 0) used += ringsize;
        if (start & (WriteSize - 1))
            writesize = WriteSize - (start & (WriteSize - 1));    /* re-align */
        else
            writesize = WriteSize;
        while (used < WriteSize) {
            if (eof) {
                done = 1;
                used = end - start;
                if (used < 0) used += ringsize;
                writesize = used;
                break;
            } else if (push > pushed + 1) {
                writesize = used;
                if (used) break;
            }
            usleep(poll_usecs);
            used = end - start;
            if (used < 0) used += ringsize;
        }
        n = ringsize - start;        /* short write at end of ring? */
        if (n < writesize) writesize = n;    /* write remainder next loop */
        if (check_block) {
            /*
             * this is mostly of academic interest
             */
            fd_set w_set;
            struct timeval timeout;
            timeout.tv_sec = 0;
            timeout.tv_usec = 0;
            FD_ZERO(&w_set);
            FD_SET(1, &w_set);
            if (select(2, NULL, &w_set, NULL, &timeout) != -1) {
                if (!FD_ISSET(1, &w_set)) {
                    would_block = 1;
```

```
                    if (yield_if_blocking) {
                        writesize = 0;    /* next iteration will try again */
                        sched_yield();
                    }
                }
            }
        }
        if (writesize > 0) {
            if (start < boundary && start + writesize >= boundary) {
                writesize = boundary - start;
            }

//              writesize = write(1, buf + start, writesize);
            kafka_produce_message(buf + start, writesize);
        }
        if (writesize == -1 && errno == EINTR) writesize = 0;
        if (writesize < 0) {
            fprintf(stderr, "%s: fatal write error: %s\n",
                    progname, strerror(errno));
            eof = 1;
            fflush(stderr);
            pthread_exit(0);
        }
        start += (start + writesize >= ringsize) ? writesize - ringsize : writesize;
        if (start == boundary) {
            if (max_files && filec >= max_files) filec = 0;
            newoutfile(odir, filec++);
            boundary = -2;
        }
        pushed = push;
    }
    if (odir) process_savefile(wfile);


    fprintf(stderr, "Flushing final messages\n");
    rd_kafka_flush(rk, 10 * 1000 /* wait for max 10 seconds */);

    gettimeofday(&KAFKA_ENDED, NULL);

    /* If the output queue is still not empty there is an issue
     * with producing messages to the clusters. */
    if (rd_kafka_outq_len(rk) > 0)
        fprintf(stderr, "%% %d message(s) were not delivered\n", rd_kafka_outq_len(rk));

```

```c
    /* Destroy kafka producer instance */
    rd_kafka_destroy(rk);

    pthread_exit(NULL);
}

void usage() {
    fprintf(stderr,
            "\n"
            "Usage: %s [--help | options]\n"
            "    --help\tprints this usage summary\n"
            "    supported options include:\n"
            #ifdef JUSTCOPY
            "    (This binary was compiled with JUSTCOPY so some options are unavailable)\n"
            #else  /* JUSTCOPY */
            "      -d\tdecapsulate Cisco ERSPAN GRE packets (sets -f value)\n"
            "      -f \"...\"\tspecify a pcap filter - see manpage and -d\n"
            "      -i eth#|-\tspecify ethernet capture interface or '-' for stdin\n"
            "      -s #\tspecify packet capture \"snapshot\" length limit\n"
            "      -F\tskip the interface type (Ethernet) check\n"
            #endif /* JUSTCOPY */
            "      -r #\tspecify ring buffer size in megabytes (1-1024)\n"
            "      -c\tjust buffer stdin to stdout (works with arbitrary data)\n"
            "      -x\trequest exclusive lock (to be the only instance running)\n"
            "      -X\trun even when locking would forbid it\n"
            "      -v\tprint program version and exit\n"
            "      -Vx...x\tdisplay packet loss and buffer use - see manpage\n"
            "      -p #\tspecify full/empty polling interval in microseconds\n"
            "      -q\tsuppress buffer full warnings\n"
            "      -z #\tspecify write blocksize (even power of 2, default 65536)\n"
            "    for long-term capture\n"
            "      -o dir\tredirect pcap output to a collection of files in dir\n"
            "      -n name\tfilename (default: pcap)\n"
            "      -t\tappend a timestamp to the filename\n"
            "      -C #\tlimit each pcap file in -o dir to # times the (-r #) size\n"
            "      -G #\trotates the pcap file every # seconds\n"
            "      -W #\toverwrite pcap files in -o dir rather than start #+1 (max_files)\n"
            "      -Z postrotate-command\trun 'command file' after each rotation\n"
            "    and some of academic interest only:\n"
            "      -B\tcheck if select(2) would ever have blocked on write\n"
            "      -Y\tavoid writes which would block\n"
            "\n", progname);
}
```

```c
void kafka_produce_message(char *message, int length) {
    rd_kafka_resp_err_t err;

    retry:
    err = rd_kafka_producev(
            /* Producer handle */
            rk,
            /* Topic name */
            RD_KAFKA_V_TOPIC(KAFKA_TOPIC),
            /* Make a copy of the payload. */
            RD_KAFKA_V_MSGFLAGS(RD_KAFKA_MSG_F_COPY),
            /* Message value and length */
            RD_KAFKA_V_VALUE(message, length),
            /* Per-Message opaque, provided in
             * delivery report callback as
             * msg_opaque. */
            RD_KAFKA_V_OPAQUE(NULL),
            /* End sentinel */
            RD_KAFKA_V_END);

    if (err) {
        /*
         * Failed to *enqueue* message for producing.
         */
        fprintf(stderr, "%% Failed to produce to topic %s: %s\n", KAFKA_TOPIC, rd_kafka_err2str(err));

        if (err == RD_KAFKA_RESP_ERR__QUEUE_FULL) {
            fprintf(stderr, "Local queue full, flushing messages and trying again...\n");
            rd_kafka_poll(rk, 500/*block for max 1000ms*/);
            goto retry;
        }
    }

    rd_kafka_poll(rk, 0/*non-blocking*/);
}

/*
 * This thread starts the other two and then wakes every half second
 * to increment a variable the writer uses to decide if it should flush.
 * Flushing greatly facilitates interactive use and testing tcpdump filters.
 */
int main(int argc, char *argv[], char *envp[]) {

    /* log purpose */
```

```c
    raw_time = time(NULL);
    time(&raw_time);
    date_info = localtime(&raw_time);

    // kafka init
    char error_buffer[512];
    rd_kafka_conf_t *conf;
    conf = rd_kafka_conf_new();

    if (rd_kafka_conf_set(conf, "bootstrap.servers", KAFKA_BROKERS, error_buffer, sizeof(error_buffer)) !=
        RD_KAFKA_CONF_OK) {
        fprintf(stderr, "Kafka config error: %s\n", error_buffer);
        exit(1);
    }

    rd_kafka_conf_set(conf, "security.protocol", KAFKA_SECURE_PROTOCOL, error_buffer, sizeof(error_buffer));
    rd_kafka_conf_set(conf, "ssl.certificate.location", CERTIFICATE_LOCATION, error_buffer, sizeof(error_buffer));
    rd_kafka_conf_set(conf, "ssl.key.location", CERTIFICATE_KEY_LOCATION, error_buffer, sizeof(error_buffer));
    rd_kafka_conf_set(conf, "ssl.ca.location", CA_CERTIFICATE_LOCATION, error_buffer, sizeof(error_buffer));

    rk = rd_kafka_new(RD_KAFKA_PRODUCER, conf, error_buffer, sizeof(error_buffer));
    if (!rk) {
        fprintf(stderr,
                "%% Failed to create kafka producer: %s\n", error_buffer);
        exit(1);
    }

    pthread_t threads[2];
    int rc, t, c, errflag = 0;
    extern char *optarg;
    extern int optind;
    int bitmask;

    start = end = eof = 0;
    progname = argv[0];
#ifdef JUSTCOPY
    just_copy = 1;
#endif

    /* pick up default interface to sniff from ENV if present */
```

```c
    if (getenv("CAP_IFACE"))
        dev = getenv("CAP_IFACE");

    if (argc > 1 && strcmp(argv[1], "--help") == 0)
        ++errflag;
    else
#ifndef JUSTCOPY
        while ((c = getopt(argc, argv, "BFXYcdqtvxf:i:p:r:s:z:V:o:n:C:G:W:Z:")) != EOF)
#else  /* JUSTCOPY */
            while ((c = getopt(argc, argv, "BXYcqtvxp:r:z:V:o:n:C:G:W:Z:")) != EOF)
#endif /* JUSTCOPY */
        {
            switch (c) {
                case 'B':
                    check_block = 1;    /* use select to avoid write blocking */
                    break;
                case 'F':
                    check_eth = 0;            /* don't check that we are capturing from an */
                    break;                  /* Ethernet device */
                case 'Y':
                    check_block = 1;
                    yield_if_blocking = 1;    /* don't issue blocking writes */
                    break;
                case 'V':            /* produce periodic drop,ring stats */
                    ps_stat_ptr = optarg;
                    if (ps_stat_ptr[0] == '-') {
                        fprintf(stderr, "%s: %s is suspicious as argument of -V\n",
                                progname, ps_stat_ptr);
                        errflag++;
                    }
                    ps_stat_len = strlen(ps_stat_ptr);
                    break;
                case 'c':
                    just_copy = 1;        /* just read from stdin and buffer */
                    break;
                case 'd':
                    gre_hdrlen = GRE_HDRLEN;/* decapsulate Cisco GRE */
                    filter_exp = "proto gre";
                    break;
                case 'f':
                    filter_exp = optarg;
                    break;
                case 'i':            /* specify ethernet device name */
                    dev = optarg;
```

```
                break;
        case 'p':            /* specify polling sleep u_secs */
            t = atoi(optarg);
            if (t < 0 || t > 1000000) {
                fprintf(stderr, "%s: -p number must be 0-1000000\n",
                        progname);
                ++errflag;
            } else poll_usecs = t;
            break;
        case 'q':            /* warnings can be annoying */
            warn_buf_full = 0;
            break;
        case 'r':
            t = atoi(optarg);    /* specify ring size in MB */
            if (t < 1 || t > 1024) {
                fprintf(stderr, "%s: -r number must be 1-1024\n",
                        progname);
                ++errflag;
            } else ringsize = t * 1024 * 1024;
            break;
        case 's':            /* specify snapshot length */
            t = atoi(optarg);
            if (t <= 0 || t > SNAP_LEN) t = SNAP_LEN;
            snap_len = t;
            break;
        case 'v':
            fprintf(stderr, "%s\n", id + 5);
            exit(0);
            break;
        case 'x':
            xlock = 1;        /* request exclusive lock */
            break;
        case 'X':
            xlock = -1;        /* disregard locking conflicts */
            break;
        case 'z':
            t = atoi(optarg);    /* specify goal write size 2^n */
            // default on for condition was 65536. why? is limited?
            for (bitmask = 1; bitmask <= 1048576; bitmask *= 2) {
                if (t == bitmask)
                    WriteSize = t;
            }
            if (WriteSize != t) {
                fprintf(stderr, "%s: -z number must be a power of 2\n",
```

```c
                        progname);
                errflag++;
            }
            break;
        case 't':
            tflag = 1;
            break;
        case 'n':
            oname = optarg;
            break;
        case 'o':
            odir = optarg;
            if (strlen(odir) >= PATH_MAX - strlen(TEMPLATE) - 1) {
                fprintf(stderr, "%s: -o name too long: %s\n",
                        progname, odir);
                errflag++;
            }
            break;
        case 'C':
            split_after = atoi(optarg);
            if (split_after < 1) {
                fprintf(stderr, "%s: -C # must be 1 or greater\n",
                        progname);
                errflag++;
            }
            break;
        case 'G':
            split_seconds = atoi(optarg);
            if (split_seconds < 1) {
                fprintf(stderr, "%s: -G # must be 1 or greater\n",
                        progname);
                errflag++;
            }
            break;
        case 'W':
            max_files = atoi(optarg);
            if (max_files < 1) {
                fprintf(stderr, "%s: -W # must be 1 or greater\n",
                        progname);
                errflag++;
            }
            break;
        case 'Z':
            zcmd = optarg;
```

```
                zflag = 1;
                break;
            default:
                errflag++;
                break;
        }
    }
    if (errflag || optind < argc) {
        usage();
        exit(1);
    }

    /* to avoid zombies when using -Z */
    (void) sigset(SIGCHLD, child_cleanup);

    /*
     * if -d is spcified, -s refers to decapsulated sizes, make it happen
     */
    d_snap_len = snap_len + gre_hdrlen;
    if (d_snap_len <= 0 || d_snap_len > SNAP_LEN)
        d_snap_len = SNAP_LEN;

    /*
     * Advisory locking logic
     */
    if ((lockfd = open("/proc/self/exe", O_RDONLY)) < 0) {
        fprintf(stderr, "%s: Warning: couldn't open lockfile so not locking\n",
                progname);
    } else {
        if (flock(lockfd, ((xlock == 1 ? LOCK_EX : LOCK_SH) | LOCK_NB)) == -1) {
            if (xlock < 0) {
                fprintf(stderr, "%s: Warning: overriding locking\n",
                        progname);
            } else {
                fprintf(stderr, "%s: Exiting due to lock conflict\n",
                        progname);
                exit(1);
            }
        }
    }

    buf = malloc(ringsize + 1);
    if (!buf) {
        fprintf(stderr, "%s: Malloc failed, exiting\n", progname);
```

```
        exit(1);
    }

    if (mlock(buf, ringsize + 1) != 0) {
        fprintf(stderr, "%s: Warning: could not lock ring buffer into RAM\n",
                progname);
    }

    rc = pthread_create(&threads[0], NULL, &Reader, NULL);
    if (rc) {
        fprintf(stderr, "%s: pthread_create error\n", progname);
        exit(1);
    }

    rc = pthread_create(&threads[1], NULL, &Writer, NULL);
    if (rc) {
        fprintf(stderr, "%s: pthread_create error\n", progname);
        exit(1);
    }

    while (!eof) {
        usleep(500000);
        push += 1;
        /*
         * emit some stats which may be useful while testing
         * if argument to -V is big enough to write into, do so
         * else write to stdout.
         */
        if (ps_stat_ptr) {
            char sbuf[V_WIDTH + 1];
            int drop_symb = 0;
            int used = end - start;
            if (used < 0) used += ringsize;
#ifndef JUSTCOPY
            if (handle && pcap_stats(handle, &pcs) >= 0) {
                int d = pcs.ps_drop;
                /* count how many decimal digits are in the drop count */
                for (drop_symb = 0; drop_symb < 9; ++drop_symb) {
                    if (d == 0) break;
                    d /= 10;
                }
            }
#endif /* JUSTCOPY */
            if (ps_stat_len >= V_WIDTH) {    /* put stats in arg list */
```

```
                sprintf(sbuf, "%1.1d %.01f,%.01f%%",
                        drop_symb,    /* a digit from 0-9 */
                        100.0 * (double) used / (double) (ringsize),
                        100.0 * (double) maxbuffered / (double) (ringsize));
                sprintf(ps_stat_ptr, "%-*s", ps_stat_len, sbuf);
            } else {                /* puts stats on stderr */
                fprintf(stderr,
                        "pkts dropped: %d, ring buf: %.1lf%%, max: %.1lf%%\n",
                        (drop_symb > 0 ? pcs.ps_drop : 0),
                        100.0 * (double) used / (double) (ringsize),
                        100.0 * (double) maxbuffered / (double) (ringsize));
            }
        }
    }

    fflush(stderr);
    pthread_exit(NULL);
}
```

Listing E.2: Packet capture version 3 application complete source-code

```
from confluent_kafka import Producer
from configs import *
import subprocess
import threading
import datetime
import signal
import json
import time
import re
import os

PCAP_HEADER_SIZE = 24
PCAP_PACKET_HEADER_SIZE = 16

producer = Producer({
    'bootstrap.servers': BOOTSTRAP_SERVERS,
    'security.protocol': 'SSL',
    'ssl.certificate.location': CERTIFICATE_LOCATION,
    'ssl.key.location': CERTIFICATE_KEY_LOCATION,
    'ssl.ca.location': CA_LOCATION
})

```

```python
tcpdump_process = None

TCPDUMP_FINISHED = False

# logging
if not os.path.exists('logs'):
    os.mkdir('logs')

date = datetime.datetime.now()
LOG_FILE = f'logs/producer_{date.year}-{date.month}-{date.day}:' \
           f'{date.hour}_{date.minute}_{date.second}.json'

log_capture_started = None
log_capture_ended = None
log_tcpdump_packets_captured = None
log_tcpdump_packets_dropped = None
log_tcpdump_packets_received_by_filter = None
log_kafka_ended = None
log_pcap_sh1 = None


def signal_handler(sig, frame):
    if tcpdump_process is not None:
        tcpdump_process.terminate()


def start_writer():
    global tcpdump_process, TCPDUMP_FINISHED, log_capture_ended, log_capture_started, log_tcpdump_packets_captured, \
        log_tcpdump_packets_dropped, log_tcpdump_packets_received_by_filter

    log_capture_started = time.time()

    tcpdump_process = subprocess.Popen(
        ['tcpdump', TCPDUMP_FILTER, '-i', NIC, '-s', str(TCPDUMP_SNAPLEN), '-w', PCAP_FILE_NAME],
        stdout=subprocess.PIPE, stderr=subprocess.STDOUT, universal_newlines=True)

    # wait for the process to close somehow (maybe unnecessary since stdout.read waits for the output)
    tcpdump_process.wait()

    tcpdump_output = tcpdump_process.stdout.read()

    TCPDUMP_FINISHED = True

```

```
    log_capture_ended = time.time()

    # get statistics written on stdout by TCPDump
    re_result = re.findall("\n[0-9]+", str(tcpdump_output))
    log_tcpdump_packets_captured = int(re_result[0])
    log_tcpdump_packets_received_by_filter = int(re_result[1])
    log_tcpdump_packets_dropped = int(re_result[2])

    print(f"TCPDump has finished")
    print("Publishing the rest of the data...")


def start_reader():
    global log_kafka_ended

    pcap_skipped = False

    pcap_file = open(PCAP_FILE_NAME, 'rb')
    left_overs = b''

    while True:
        chunk = left_overs + pcap_file.read(
            KAFKA_MESSAGE_SIZE - len(left_overs))  # join the left overs and the new data
        left_overs = b''

        if chunk == b'' and TCPDUMP_FINISHED:
            break

        if chunk == b'':
            continue

        pointer = 0

        while len(chunk) > pointer:  # loop while we can fetch data

            if not pcap_skipped:
                chunk = chunk[PCAP_HEADER_SIZE:]  # remove the file header
                pcap_skipped = True

            if len(chunk) < pointer + PCAP_PACKET_HEADER_SIZE:  # can we not get the packet header?
                # left_overs = chunk[pointer:]
                break

            caplen = int.from_bytes(chunk[pointer + 8:pointer + 12], 'little')
```

```

            if len(chunk) < pointer + PCAP_PACKET_HEADER_SIZE + caplen:  # can we not get packet header + the packet?
                # left_overs = chunk[pointer:]
                break

            pointer += PCAP_PACKET_HEADER_SIZE + caplen  # move the pointer for the next packet header

        left_overs = chunk[pointer:]
        # we can't fetch more data, so publish the message and lets get another chunk
        try:
            if chunk[:pointer] != b'':
                producer.produce(KAFKA_TOPIC, chunk[:pointer])  # lets write the chunk, start to the pointer

        except BufferError:
            print('Local queue full, flushing messages and trying again')
            producer.flush()
            producer.produce(KAFKA_TOPIC, chunk[:pointer])
        finally:
            producer.poll(0)

    print('Flushing final messages...')
    producer.flush()
    log_kafka_ended = time.time()


# start the capture as a thread
thread_capture = threading.Thread(target=start_writer)
thread_capture.start()

# let the tcpdump start first
time.sleep(0.3)
thread_publish = threading.Thread(target=start_reader)
thread_publish.start()

signal.signal(signal.SIGINT, signal_handler)
print('Press Ctrl+C to stop the capture')
signal.pause()

thread_capture.join()
thread_publish.join()

# script ended
```

```python
log_end_time = time.time()

print(
    f'\nPackets captured: {log_tcpdump_packets_captured}\nPackets dropped: {log_tcpdump_packets_dropped}'
    f'\nPackets received by filter: {log_tcpdump_packets_received_by_filter}'
)

print('\nGenerating logs...')

capinfo = subprocess.check_output(f"capinfos {PCAP_FILE_NAME} -M", shell=True, text=True)
log_pcap_total_packets = re.findall("Number of packets: +[0-9]+", capinfo)[0]
log_pcap_total_packets = int(re.findall("[0-9]+", log_pcap_total_packets)[0])

log_pcap_average_packet_rate = re.findall("Average packet rate: +[0-9]+[.]?[0-9]*", capinfo)[0]
log_pcap_average_packet_rate = float(re.findall("[0-9]+[.]?[0-9]*", log_pcap_average_packet_rate)[0])

log_pcap_size_bytes = re.findall("File size: +[0-9]+", capinfo)[0]
log_pcap_size_bytes = int(re.findall("[0-9]+", log_pcap_size_bytes)[0])

log_pcap_data_size_bytes = re.findall("Data size: +[0-9]+", capinfo)[0]
log_pcap_data_size_bytes = int(re.findall("[0-9]+", log_pcap_data_size_bytes)[0])

log_output = {
    'method': 'file',
    'capture_started': log_capture_started,
    'capture_ended': log_capture_ended,
    'snaplen': TCPDUMP_SNAPLEN,
    'packets_captured': log_tcpdump_packets_captured,
    'packets_dropped': log_tcpdump_packets_dropped,
    'packets_received_by_filter': log_tcpdump_packets_received_by_filter,
    'kafka_ended': log_kafka_ended,
    'kafka_chunks_bytes': KAFKA_MESSAGE_SIZE,
    'pcap_packets': log_pcap_total_packets,
    'pcap_average_packet_rate_sec': log_pcap_average_packet_rate,
    'pcap_size_bytes': log_pcap_size_bytes,
    'pcap_data_size_bytes': log_pcap_data_size_bytes,
    'full_capinfo_log': capinfo
}

with open(LOG_FILE, 'w') as f:
    json.dump(log_output, f, indent=4)
```

Appendix F

Persistent network data application source code

Listing F.1: Persistent network data application complete source-code

```
from confluent_kafka import Consumer
from requests import Session
from hdfs import Client
from configs import *
import threading
import datetime
import signal
import json
import time
import sys
import os

FILE_NAME = sys.argv[1]

PCAP_GLOBAL_HEADER = b'\xd4\xc3\xb2\xa1\x02\x00\x04\x00\x00' \
                     b'\x00\x00\x00\x00\x00\x00\x00\x00\x00' \
                     b'\x04\x00\x01\x00\x00\x00'

FILE_FULL_PATH = (DIRECTORY if DIRECTORY[-1] == '/' else DIRECTORY + '/') + FILE_NAME

CONSUMING = True

# logging
```

```python
if not os.path.exists('logs'):
    os.mkdir('logs')

date = datetime.datetime.now()
LOG_FILE = f'logs/persistent_storage_{date.year}-{date.month}' \
           f'-{date.day}:{date.hour}_{date.minute}_{date.second}.json'
log_start_time = None
log_end_time = None


class SecureClient(Client):

    def __init__(self, url, cert=None, verify=True, **kwargs):
        session = Session()
        if ',' in cert:
            session.cert = [path.strip() for path in cert.split(',')]
        else:
            session.cert = cert
        session.verify = verify
        super(SecureClient, self).__init__(url, session=session, **kwargs)


client = SecureClient(HADOOP_URI, cert=PEMFILE, verify=False)


def signal_handler(sig, frame):
    global CONSUMING

    CONSUMING = False


def packets_consumer():
    global log_start_time, log_end_time

    with client.write(FILE_FULL_PATH) as writer:

        writer.write(PCAP_GLOBAL_HEADER)

        consumer = Consumer({
            'bootstrap.servers': BOOTSTRAP_SERVERS,
            'group.id': GROUP_ID,
            'auto.offset.reset': 'earliest',
            'security.protocol': 'SSL',
            'ssl.certificate.location': CERTIFICATE_LOCATION,
```

```
            'ssl.key.location': CERTIFICATE_KEY_LOCATION,
            'ssl.ca.location': CA_LOCATION
        })

        consumer.subscribe(KAFKA_TOPICS)

        log_start_time = time.time()

        while True:
            message = consumer.poll(POOL_WAIT)

            if not CONSUMING and message is None:
                break

            if message is None:
                continue

            writer.write(message.value())

        consumer.close()
        writer.flush()

    log_end_time = time.time()


thread_consumer = threading.Thread(target=packets_consumer)
thread_consumer.start()

signal.signal(signal.SIGINT, signal_handler)
print('Press Ctrl+C to stop consuming')
signal.pause()

thread_consumer.join()

log_output = {
    'start_time': log_start_time,
    'end_time': log_end_time
}

with open(LOG_FILE, 'w') as f:
    json.dump(log_output, f, indent=4)
```

Appendix G

Analysis module applications source code

Listing G.1: Information analyzer source-code

```
import threading
import datetime
import signal
import time
import json
import os
from confluent_kafka import Consumer
from ipaddress import IPv4Address, IPv6Address
from pcap import *
from configs import *

PCAP_HEADER_SIZE = 24
PCAP_PACKET_HEADER_SIZE = 16

IPV4 = b'\x08\x00'
IPV6 = b'\x86\xdd'
ARP = b'\x08\x06'
ICMP = b'\x01'
TCP = b'\x06'
UDP = b'\x11'

ETHERNET_SIZE = 14
IPV4_SIZE = 20
```

```
IPV6_SIZE = 40
ARP_SIZE = 28
TCP_SIZE = 20
UDP_SIZE = 8

PCAP_GLOBAL_HEADER_SKIPPED = True

LEFT_OVERS = b''

CONSUMING = True

TOTAL_PACKETS = 0
TCP_PACKETS = 0
UDP_PACKETS = 0
ARP_PACKETS = 0
ICMP_PACKET = 0

START_DATE = datetime.datetime.now()
START_TIME = None
END_TIME = None

# key is the ip, value is the number of requests
SOURCE_IP_ADDRESSES = {}
DESTINATION_IP_ADDRESSES = {}

if not os.path.exists('logs'):
    os.mkdir('logs')


def signal_handler(sig, frame):
    global CONSUMING

    CONSUMING = False


def kafka_consumer():
    global END_TIME, START_TIME

    consumer = Consumer({
        'bootstrap.servers': BOOTSTRAP_SERVERS,
        'group.id': GROUP_ID,
        'auto.offset.reset': 'earliest',
        'security.protocol': 'SSL',
        'ssl.certificate.location': CERTIFICATE_LOCATION,
```

```python
        'ssl.key.location': CERTIFICATE_KEY_LOCATION,
        'ssl.ca.location': CA_LOCATION
    })

    consumer.subscribe([KAFKA_TOPIC])

    START_TIME = time.time()

    while True:
        message = consumer.poll(POOL_WAIT)

        if not CONSUMING and message is None:
            break

        if message is None:
            continue

        handle_chunk(LEFT_OVERS + message.value())

    END_TIME = time.time()

    consumer.close()

    print(f'Unprocessed bytes: {len(LEFT_OVERS)}')


def handle_chunk(chunk):
    global PCAP_GLOBAL_HEADER_SKIPPED, LEFT_OVERS

    LEFT_OVERS = b''

    pointer = 0

    while pointer < len(chunk):
        if pointer + PCAP_PACKET_HEADER_SIZE > len(chunk):
            LEFT_OVERS = chunk[pointer:]
            break

        raw_header = chunk[pointer:pointer + PCAP_PACKET_HEADER_SIZE]

        caplen = int.from_bytes(raw_header[8:12], 'little')

        pointer += PCAP_PACKET_HEADER_SIZE  # advance to the actual packet

```

```
        if pointer + caplen > len(chunk):
            LEFT_OVERS = raw_header + chunk[pointer:]  # also include the pcap packet header
            break

        handle_packet(chunk[pointer:pointer + caplen])  # get the packet

        pointer += caplen


def handle_packet(packet):
    global TOTAL_PACKETS

    if len(packet) < ETHERNET_SIZE:
        print('Invalid ethernet packet')
        return

    TOTAL_PACKETS += 1

    mac_dst = packet[:6].hex(':')
    mac_src = packet[6:12].hex(':')

    protocol = packet[12:14]

    upper_layer = packet[14:]

    if protocol == IPV4 and len(upper_layer) >= IPV4_SIZE:  # sanity test
        handle_ipv4_packet(upper_layer)
    elif protocol == IPV6 and len(upper_layer) >= IPV6_SIZE:
        handle_ipv6_packet(upper_layer)
    elif protocol == ARP and len(upper_layer) >= ARP_SIZE:
        handle_arp_packet(upper_layer)
    else:
        pass  # unknown protocol or malefactor one


def handle_ipv4_packet(packet):
    ip_src = str(IPv4Address(packet[12:16]))
    ip_dst = str(IPv4Address(packet[16:20]))

    if ip_src in SOURCE_IP_ADDRESSES:
        SOURCE_IP_ADDRESSES[ip_src] = SOURCE_IP_ADDRESSES[ip_src] + 1
    else:
        SOURCE_IP_ADDRESSES[ip_src] = 1

```

```python
    if ip_dst in DESTINATION_IP_ADDRESSES:
        DESTINATION_IP_ADDRESSES[ip_dst] = DESTINATION_IP_ADDRESSES[ip_dst] + 1
    else:
        DESTINATION_IP_ADDRESSES[ip_dst] = 1

    protocol = packet[9:10]
    upper_layer = packet[IPV4_SIZE:]

    if protocol == TCP and len(upper_layer) >= TCP_SIZE:
        handle_tcp_packet(upper_layer)
    elif protocol == UDP and len(upper_layer) >= UDP_SIZE:
        handle_udp_packet(upper_layer)
    elif protocol == ICMP and len(upper_layer) >= 1:
        handle_icmp_packet(upper_layer)
    else:
        pass  # unknown protocol or malefactor one


def handle_ipv6_packet(packet):
    ip_src = str(IPv6Address(packet[8:24]))
    ip_dst = str(IPv6Address(packet[24:40]))

    if ip_src in SOURCE_IP_ADDRESSES:
        SOURCE_IP_ADDRESSES[ip_src] = SOURCE_IP_ADDRESSES[ip_src] + 1
    else:
        SOURCE_IP_ADDRESSES[ip_src] = 1

    if ip_dst in DESTINATION_IP_ADDRESSES:
        DESTINATION_IP_ADDRESSES[ip_dst] = DESTINATION_IP_ADDRESSES[ip_dst] + 1
    else:
        DESTINATION_IP_ADDRESSES[ip_dst] = 1

    protocol = packet[6:7]
    upper_layer = packet[IPV6_SIZE:]

    if protocol == TCP and len(upper_layer) >= TCP_SIZE:
        handle_tcp_packet(upper_layer)
    elif protocol == UDP and len(upper_layer) >= UDP_SIZE:
        handle_udp_packet(upper_layer)
    elif protocol == ICMP and len(upper_layer) >= 1:
        handle_icmp_packet(upper_layer)
    else:
        pass  # unknown protocol or malefactor one

```

```python

def handle_arp_packet(packet):
    global ARP_PACKETS

    ARP_PACKETS += 1

    # print('arp packet')


def handle_tcp_packet(packet):
    global TCP_PACKETS

    TCP_PACKETS += 1

    # print('tcp packet')

    port_src = int.from_bytes(packet[:2], 'big')
    port_dst = int.from_bytes(packet[2:4], 'big')


def handle_udp_packet(packet):
    global UDP_PACKETS

    UDP_PACKETS += 1

    # print('udp packet')

    port_src = int.from_bytes(packet[:2], 'big')
    port_dst = int.from_bytes(packet[2:4], 'big')


def handle_icmp_packet(packet):
    global ICMP_PACKET

    ICMP_PACKET += 1

    # print('icmp packet')


def reporter():
    last_packets_count = TOTAL_PACKETS

    while CONSUMING:
        time.sleep(REPORTER_DELAY)
```

```python

        # print only if new data has arrived
        if last_packets_count >= TOTAL_PACKETS:
            continue

        last_packets_count = TOTAL_PACKETS

        print(f'TCP packets: {TCP_PACKETS}, UDP packets: {UDP_PACKETS}, ARP packets {ARP_PACKETS}, '
              f'ICMP packets {ICMP_PACKET}')

        if SOURCE_IP_ADDRESSES:
            top1_source_ip = max(SOURCE_IP_ADDRESSES, key=SOURCE_IP_ADDRESSES.get)
            print(f'Top source IP: {top1_source_ip} with {SOURCE_IP_ADDRESSES[top1_source_ip]} requests')

        if DESTINATION_IP_ADDRESSES:
            top1_destination_ip = max(DESTINATION_IP_ADDRESSES, key=DESTINATION_IP_ADDRESSES.get)
            print(
                f'Top destination IP: {top1_destination_ip} with
                ↪  {DESTINATION_IP_ADDRESSES[top1_destination_ip]} '
                f'requests\n\n')


if __name__ == '__main__':
    thread_consumer = threading.Thread(target=kafka_consumer)
    thread_consumer.start()

    thread_reporter = threading.Thread(target=reporter)
    thread_reporter.start()

    signal.signal(signal.SIGINT, signal_handler)
    print('Press Ctrl+C to stop consuming')
    signal.pause()
    thread_consumer.join()
    thread_reporter.join()

    # kafka_consumer()
    print(f'Packets analyzed: {TOTAL_PACKETS}')
    print(f'TCP packets: {TCP_PACKETS}, UDP packets: {UDP_PACKETS}, ARP packets {ARP_PACKETS}, '
          f'ICMP packets {ICMP_PACKET}')

    log_output = {
        'start_time': START_TIME,
        'end_time': END_TIME,
        'total_packets': TOTAL_PACKETS
```

```
    }

    LOG_FILE = f'logs/custom_parser_analyzer_{START_DATE.year}-' \
               f'{START_DATE.month}-{START_DATE.day}:{START_DATE.hour}' \
               f'_{START_DATE.minute}_' \
               f'{START_DATE.second}.json'

    with open(LOG_FILE, 'w') as f:
        json.dump(log_output, f, indent=4)
```

Listing G.2: Dataset preperation source-code

```
import pandas as pd
import numpy as np


def load_data():
    domains = pd.read_csv('datasets/domains.csv')
    domains.drop(['RootObject.subclass'], axis=1, inplace=True)
    columns = {'RootObject.class': 'pred', 'RootObject.domain': 'domain'}
    domains.rename(columns=columns, inplace=True)

    for i in range(domains.shape[0]):
        if domains['pred'][i] == 'legit':
            domains['pred'][i] = 0
        else:
            domains['pred'][i] = 1

    return domains[['domain', 'pred']]


def strip(domain_name):
    domain_name = domain_name.lower()
    name_chunks = domain_name.split('.')

    if len(name_chunks) == 1:
        return domain_name
    else:
        return name_chunks[-2]


def preprocess(data):
    df_dict = data.to_dict('records')
```

```
    for row in df_dict:
        row['domain'] = strip(row['domain'])

    new_data = pd.DataFrame.from_dict(df_dict)

    # drop duplicates and return
    return new_data.drop_duplicates(subset=['domain'])


domains = load_data().sample(frac=1)
domains['domain'] = domains['domain'].astype(str)

domains_2 = pd.read_csv('datasets/dga.txt', index_col=False, names=['junk', 'domain', 'junk1', 'junk2'],
                        skiprows=15)
domains_2 = domains_2.drop(['junk', 'junk1', 'junk2'], axis=1)
domains_2['domain'] = domains_2['domain'].astype(str)

domains_3 = pd.read_csv('datasets/top-1m.csv', names=['domain'], index_col=0).reset_index(drop=True)
domains_3['domain'] = domains_3['domain'].astype(str)

pred_2 = np.ones(domains_2.shape[0], dtype=int)
pred_3 = np.zeros(domains_3.shape[0], dtype=int)

domains_2['pred'] = pred_2
domains_3['pred'] = pred_3

domain_data = pd.concat([domains, domains_2, domains_3], ignore_index=True, sort=True)

domain_data = preprocess(domain_data)

domain_data = domain_data.sample(frac=1).reset_index(drop=True)
domain_data.to_csv('datasets/domain_data.csv', index=False)
```

Listing G.3: Model training source-code

```
from tensorflow.keras.layers import Input, LSTM, Dropout, Embedding, Dense, Conv1D, MaxPooling1D
import tensorflow as tf
import pandas as pd
import numpy as np

EPOCHS = 6
TESTING_PERCENTAGE = 10  # 0-100
```

```python

# the max length a label can have in the domain (https://www.rfc-editor.org/rfc/rfc1035)
MAX_DOMAIN_LENGTH = 63

char2idx = {'-': 0, '.': 1, '0': 2, '1': 3, '2': 4, '3': 5,
            '4': 6, '5': 7, '6': 8, '7': 9, '8': 10, '9': 11,
            '_': 12, 'a': 13, 'b': 14, 'c': 15, 'd': 16, 'e': 17,
            'f': 18, 'g': 19, 'h': 20, 'i': 21, 'j': 22, 'k': 23,
            'l': 24, 'm': 25, 'n': 26, 'o': 27, 'p': 28, 'q': 29,
            'r': 30, 's': 31, 't': 32, 'u': 33, 'v': 34, 'w': 35,
            'x': 36, 'y': 37, 'z': 38}


def load_tf_dataset(domains):
    lines = []
    for i, line in enumerate(domains.iloc[:, 0]):
        lines.append([char2idx[c] for c in line])

    # pad the rest with 0 so they all have the same length
    tensor = tf.keras.preprocessing.sequence.pad_sequences(lines, maxlen=MAX_DOMAIN_LENGTH, padding='post')
    targets = np.array(domains.iloc[:, 1], dtype=np.int32)

    data = tf.data.Dataset.from_tensor_slices(tensor)
    pred = tf.data.Dataset.from_tensor_slices(targets)
    dataset = tf.data.Dataset.zip((data, pred))

    return dataset


def create_model():
    domain_input = Input(shape=(MAX_DOMAIN_LENGTH,), dtype='int32', name='domain_input')
    embedding = Embedding(input_dim=39, output_dim=128, input_length=MAX_DOMAIN_LENGTH,
                          batch_input_shape=[1500, None])(domain_input)
    conv = Conv1D(filters=128, kernel_size=3, padding='same', activation='relu', strides=1)(embedding)
    pool = MaxPooling1D(pool_size=2, padding='same')(conv)
    lstm = LSTM(64, return_sequences=False)(pool)
    drop = Dropout(0.5)(lstm)
    output = Dense(1, activation='sigmoid')(drop)
    model = tf.keras.Model(inputs=domain_input, outputs=output)
    return model


domains = pd.read_csv('datasets/domain_data.csv', dtype={0: str}, keep_default_na=False)

```

```
training_percentage = 100 - TESTING_PERCENTAGE
training_range = int(len(domains) * (training_percentage / 100))

dataset = load_tf_dataset(domains[:training_range])
test_dataset = load_tf_dataset(domains[training_range:])

dataset = dataset.batch(1500, drop_remainder=True)
test_dataset = test_dataset.batch(1500, drop_remainder=True)

# building model
model = create_model()
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

accuracy = []
losses = []

for i in range(EPOCHS):
    history = model.fit(dataset)

    accuracy.append(history.history['accuracy'])
    losses.append(history.history['loss'])

print('accuracy between epochs')
print(accuracy)
print('losses between epochs')
print(losses)

model.evaluate(test_dataset)
model.save('models/dga_classifier.h5')
```