POLITECNICO DI TORINO

Dipartimento di Elettronica e Telecomunicazioni
Corso di Laurea Magistrale in Communications and Computer Networks Engineering

Tesi di Laurea Magistrale

DPIpot - Analysis of Anomalous Traffic Via DPI Enhanced Honeypots

Supervisors:
Prof. Marco Mellia
Prof. Idilio Drago

Candidate:
Tommaso Rescio
S259578

April 2021

Contents

List of Figures

List of Tables

Abstract

The exponential spread of the Internet over the last decades carries a simultaneous increase and sharpening of cybersecurity attacks. Given that the current information society is almost totally dependent on technological solutions, cyber-attacks generate truly disastrous repercussions on human relationships, on work activities, on scientific progress and on economic growth. Bearing in mind these aspects, in this thesis we contribute to the development of automatic methods for monitoring the network and identifying cyber-threats.

For this purpose, we implement a flexible set of honeypots, i.e., decoys that mimic a target for hackers and use their intrusion attempts to gain information about them. More in detail, we consider a flexible infrastructure, whose configuration can be changed dynamically to select which kind of services to expose, on which ports, how to reply, etc. We analyze the impact that different deployments can have on the type of information obtained about attacks. In particular, the goal is to understand a) what is the share of traffic reaching the honeypots that arrives at different attack phases (i.e., just probing open ports, establishing the three-way handshake, actually sending packets or performing login attempts), b) whether the attack pattern changes, when we start replying to all the connection requests, c) whether the attack pattern changes depending on the kind of services we expose, d) whether by identifying protocols on-the-fly before replying, even when traffic reaches non-standard ports, influences the attack patterns. In order to identify protocols on-the-fly, we propose a novel solution called DPIpot, an infrastructure that is able to accept requests on all ports and to recognize the attacker's protocol through DPI (Deep Packet Inspection) and to address the input traffic to the most appropriate protocol-specific honeypot. To understand the changes in the traffic hitting each scenario, we use Darknet traffic as a baseline. A Darknet is a network composed of IP addresses that are publicly reachable but do not provide nor host any service. The addresses in the Darknet, as opposed to the honeypot ones, passively record the received packets, without answering any request. By comparing these two ecosystems, we are able to spot

and highlight the most significant differences, and gain a wider knowledge of the attackers' behavior.

As a main result of the DPI analysis, we observed that all tools reach steady-state classification after one packet, suggesting they can be exploited in online scenarios. Indeed, we choose nDPI as the best library to perform Deep Packet inspection on-the-fly since it is the most accurate - with respect to the other considered DPI tools - and it provides a good trade-off between accuracy and number of supported protocols.

As a main result of the traffic analysis, we record an increment in traffic when we start replying: this increment is visible both in traffic volume and the number of remote sources reaching the infrastructure. We then observe that often, when the attacker finds an active service, it immediately starts probing adjacent ports. When we expose only some services, we observe a significant increase only for those services. Furthermore, in most of the cases, there is no significant difference between replying at level four only or at layer seven only, except for some specific services, like RDP. Thus, in many cases, completing the three-way handshake is enough to engage more attackers. The analysis of the traffic reaching DPIpot leads to other interesting observations. We record that attacks on non-standard ports are very common, and replying with the correct protocol attracts a larger volume of attacks. For some services, we see that the largest part of the traffic is directed to non-standard ports.

Chapter 1

Introduction

The exponential spread of the Internet over the last few years carries a simultaneous increase and sharpening of cybersecurity attacks. Given that the *information society* is almost totally dependent on technological solutions, cyber-attacks can generate truly disastrous repercussions on human relationships, work activities, scientific progress and economic growth. It is enough to think to our everyday life activities, the frequent use of smart working in the most varied working sectors, the unstoppable expansion of social media, and the use of information technology in most of the production activities, to understand that a disservice caused by a cyberattack can damage not only human rights but also the very functioning of society. For example, in November 2016, more than 900,000 Deutsche Telekom routers went offline after being infected with a variant of Mirai[1], a malware designed to operate on Internet-connected devices, especially IoT devices, making them part of a botnet that can be used for large-scale cyber attacks. In October 2013, Adobe reported that hackers had stolen IDs and encrypted passwords for 38 million active users: Adobe has forced to pay 1 million in legal fees and an undisclosed amount to users to settle claims of violating the Customer Records Act and unfair business practice [2]. Therefore, one of the main objectives of IT security is the search for strategies capable of preventing the risk of threats [1]. More specifically, the goal is to preserve confidentiality (the concealment of information or

[1] https://krebsonsecurity.com/2016/11/new-mirai-worm-knocks-900k-germans-offline/

[2] https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html

resources), integrity (the trustworthiness of data or resources, i.e. preventing improper or unauthorized change) and availability (the ability to use the information or resource desired) of any digital and information technologies [2].

The collection and analysis of accurate, concise, and high-quality information about malicious activities is especially helpful in gaining insights about zero-day threats and newly born botnets. Since attackers are much faster in exploiting the vulnerabilities than vendors are in creating and rolling out patches [3], relying only on techniques such as Intrusion Detection Systems, Intrusion Prevention Systems, antivirus and dynamic firewalls is nowadays not enough. It is indeed necessary to resort to automatic anomaly identification tools.

In this thesis, we present a honeypot infrastructure able to contribute to the recognition of new threats and botnets, as well as new attack patterns. We analyze how the traffic hitting the infrastructure changes when we deploy different sets of honeypots exposing different vulnerable services. Honeypots are systems that are intentionally left vulnerable, with the final objective of luring the attacker to gain a larger insight into his activities. We characterize the changes in traffic using as a reference baseline the unwanted traffic hitting our Darknet, a set of advertised IP addresses that capture these packets without answering or performing any other active request.

1.1 Motivation

Since the Internet traffic is constantly growing, also the volume of cyberattacks hitting the networks is continuously increasing and adapting to exploit new vulnerabilities. For this reason, advanced and automatic methods to detect and prevent them are necessary. The Cisco Annual Internet Report[3] confirms such trends in the traffic growth, together with the need for more sophisticated countermeasures against cyber attackers:

- there will be 5.3 billion total Internet users (66 per cent of the global population) by 2023, up from 3.9 billion (51 per cent of the global population) in 2018;
- the number of devices connected to IP networks will be more than three times the global population by 2023;

[3] https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

- there will be 29.3 billion networked devices by 2023, up from 18.4 billion in 2018;
- the total number of global mobile subscribers will grow from 5.1 billion (66 per cent of the population) in 2018 to 5.7 billion (71 per cent of the population) by 2023;
- the total number of DDoS attacks will double from 7.9 million in 2018 to 15.4 million by 2023;
- there was a 776% growth in attacks between 100 Gbps and 400 Gbps from 2018 to 2019.

It is clear that a manual inspection of the traffic is not enough anymore, but it is necessary to develop automatic methods of network monitoring and identification of anomalies that are able to highlight cyber-attacks. Darknets, defined as sets of IP addresses that are advertised without answering any traffic, are one of the tools used to collect unwanted traffic, to understand the events on the network. By analysing such traffic, it is indeed possible to find out not only possible attacks but also anomalies and failures. However, a clear drawback of Darknets is that they do not reply to any request, therefore it is difficult to understand the exact behaviour of a possible attack. Thus, the main idea of this thesis is to start replying to this type of traffic to broaden our view on the characterization and understanding of such traffic. For this purpose, we use honeypots, i.e., systems that mimic a target for hackers and use their intrusion attempts to gain information about them. This is a challenging task since state-of-the-art honeypots are not flexible: they mostly present a vertical architecture, i.e., they are capable of reproducing a specific service protocol, often only if linked to a specific port. For this reason, we firstly develop a smart meta-honeypot that is able not only to accept requests on all ports but also to recognize the attacker's protocol through DPI (Deep Packet Inspection) and to address the input traffic to the most appropriate protocol-specific honeypot.

It is crucial to extend the existing honeypot environment with more flexible solutions, as the mapping protocol-port is not univocal anymore. We can think for instance of a strategy used in cybersecurity called "security through obscurity": the main idea is to implement secrecy as the main method of providing security to a system or component. For example, using port 8080 instead of port 80 for web traffic. For this reason,

it is not uncommon to observe attacks on nonstandard ports, still hosting well-known services. This is one of the reasons why we need to create a flexible and scalable framework in which more honeypots collaborate with the intent of luring the attacker for longer periods, and therefore gaining as much information as possible about attacks.

1.2 Research questions

In this thesis, we want to understand the impact of different deployments on the type of information obtained about attacks. Here we investigate whether these possible variations impact the type of information obtained by the honeypots. More in detail, we answer the following questions:

- What is the share of the traffic reaching the honeypots that arrives at different attack phases (i.e., just probing open ports, establishing the three-way handshake, actually sending packets or performing login attempts)? Which is the percentage of traffic not opening a connection?
- Does the attack pattern change if we start replying to all the connection requests?
- Does the attack pattern change depending on the kind of services we expose?
- Does identifying protocols on-the-fly before replying, even when traffic reaches non-standard ports, influence the attack patterns?

1.3 Methodology

Our methodology is based on different steps, as we see in Fig. 3.1:

1. *Analysis of DPI tools*, to choose the best performing DPI tool in terms of both accuracy (how much the library is able to recognize a specific protocol without errors) and efficiency (which is the minimum number of packets that the library needs to provide the correct protocol).
2. *Analysis of existing state-of-the-art honeypots* to identify the most up-to-date ones and consequently choose those that cover the largest number of services;

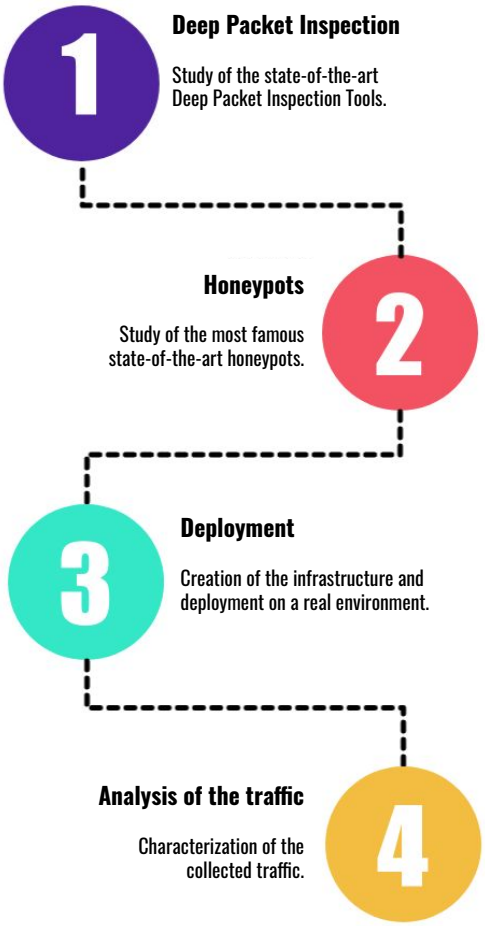


Figure 1.1: Steps.

3. *Deployment* to create the IT infrastructure and expose it to remote traffic;

4. *Analysis of the collected traffic*: we provide a breakdown analysis of the collected traffic intending to understand a) if there is an increase in the volume of attacks, b) if there are common patterns of the attacks, c) if a specific vulnerability affects possible subsequent attacks.

In particular, as it is possible to see in Fig. 1.2, starting from the left we observe the following entities: the attackers, the orchestrator, the DPIpot and the honeypots and Darknets. The unwanted traffic reaches our infrastructure, the orchestrator is in

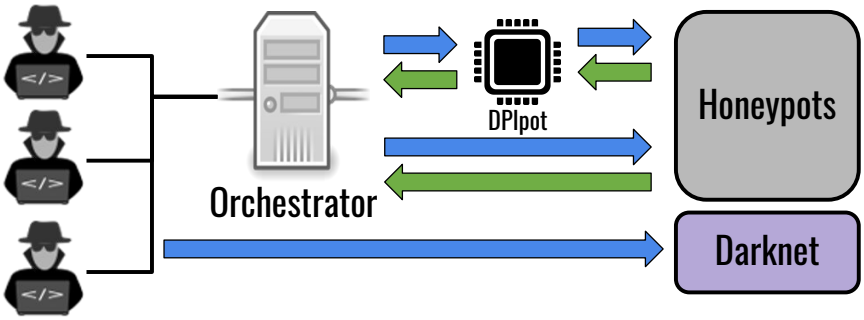


Figure 1.2: Complete framework.

charge of redirecting the packets to the desired honeypot according to some specific rules. We consider different scenarios, i.e., different configurations and deployments of honeypots, and we compare the traffic hitting our variate deployments with the one hitting the Darknet with the aim of spotting differences between the two systems. We deploy different scenarios, depending on which honeypots are involved and on which ports they are placed. We consider for example honeypot exposing Web services, DB services, Windows Remote Desktop services, Terminal services, just to name a few, and comparing everything against the baseline traffic reaching the Darknet. For each scenario we evaluate if there is an impact in volume of traffic and if there exist new common attack patterns by comparing two different situations: a) we reply only at the transport layer with an L4-Responder, b) we reply also at the application layer with an L7-Responder. The former establishes TCP connections by completing the sole three-way handshake with the probing part, while for the latter task we exploit a set of state-of-the-art honeypots. For example, when we reply at the application layer, we use T-Pot[4], the most famous and updated honeypot project. At last, we consider an additional scenario, in which through our solution, that we named DPIpot, we are able to identify on-the-fly the requested service and forward the packet to the most suitable honeypot. The DPIpot is an infrastructure, developed in this thesis, that is able to accept requests on all ports, recognize the attacker's protocol through DPI (Deep Packet Inspection) and forward it to the most appropriate protocol-specific honeypot.

[4] `https://github.com/telekom-security/tpotce`

1.4 Organization of the thesis

This thesis work is organized as follows: in Chapter 2 we present the state-of-the-art of DPI tools and honeypots, in Chapter 3 we present the evaluation of the DPI tools, in Chapter 4 we present the implementation of the DPIpot, in Chapter 5 we present the characterization of the captured traffic and in Chapter 6 we present the conclusions and the future works.

Chapter 2

State of the art

In this Chapter, we enumerate and give more details on some of the previously developed solutions in the fields of Deep Packet Inspection and Honeypot development, respectively. The final aim is to acquire a broad view of the state-of-the-art, to choose the most suitable set of tools to support our analysis.

2.1 Deep Packet Inspection

DPI has been applied to protocol identification since the early 2000s when the usage of well-known ports for traffic identification turned out to be unreliable. Multiple approaches have been proposed: some works rely on "shallow" packet inspection [4], i.e., they parse only packet headers in the search for protocol fingerprints. Such techniques still find practical applications, as encryption protects protocol payloads. Others propose efficient approaches for DPI, e.g., using pattern matching [5] or finely-tailored DPI algorithms [6]. Finally, some works rely on stateful information from multiple flows to label traffic, e.g., leveraging the DNS to obtain the labels used to classify encrypted traffic [7]. Many DPI tools have been introduced implementing such techniques. Here we evaluate four alternatives, which have been evaluated by original authors in [8, 9, 10, 11]. In contrast to them, we perform an independent evaluation of the tools, thus providing also a validation of the authors' results. Past works compare DPI solutions. Authors of [12] perform an extensive benchmark covering port-based classification, packet signature algorithms etc. In [13], authors survey ap-

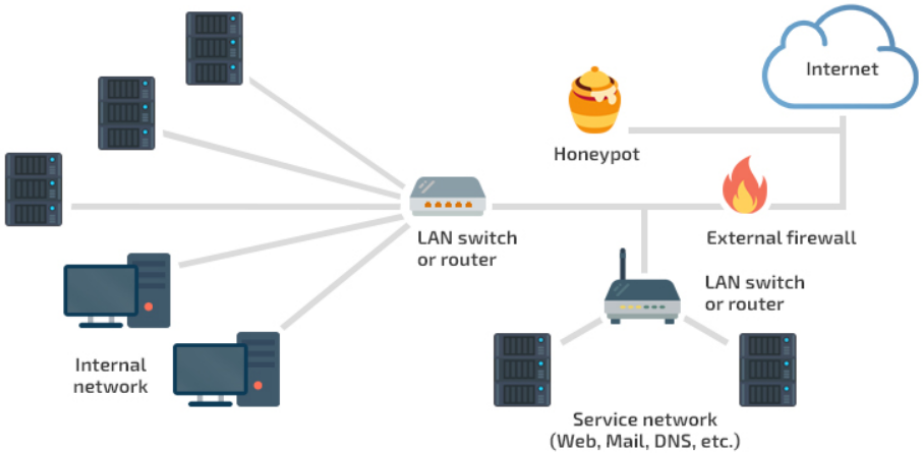


Figure 2.1: Example of a network with an honeypot.

proaches to overcome the lack of ground truth in such studies. In some cases, manual labelling of packet captures is used for DPI comparisons [14], while other works rely on active measurements to enrich captures with information about underneath applications [15, 16, 17]. Closer to our analysis is the work presented in [18], where authors also provide an independent comparison of DPI solutions. In contrast to [18], we leave out of our evaluation proprietary tools and libraries, since the lack of source code makes it hard to explore and explain discrepant results. We also refrain from evaluating tools no longer maintained. More importantly, we provide an updated comparison of DPI tools considering recent and real traces, thus covering scenarios not evaluated in the previous work, with a particular focus on modern security applications. In particular, since the goal is to use a DPI tool in a real-time scenario, we provide also an evaluation based also on how many packets needs a tool to recognize the Layer 7 protocol and whether the tool is suitable for on-the-fly classification.

2.2 Honeypots

According to [19], a honeypot is "*a security resource whose value lies in being probed, attacked, or compromised*". In essence, it is a resource that has no production value or authorized activity and therefore all connections to the honeypot are suspicious by nature: they are most likely probes, scans, or attacks. Therefore, the main advantage

of analysing the traffic collected by a honeypot is that the traffic is not influenced by false positives and negatives: although the collected traffic may be little (compared to a real production system), the majority of this traffic may be malicious. Thus, the general objective of a honeypot is twofold: first, to distract the attackers from their target, second, and more importantly, to gather information about attack patterns.

Nowadays, the main security mechanisms mostly used are: prevention (that involves mechanisms that make the attack fail), detection (that is the process of identifying the presence of an attack), or reaction (that refers to the execution of focused actions after the discovery of the attack). Example of prevention systems may be firewalls and the implementation of guidelines of cyber-security standards, for example, a recently and constantly updated clean configuration. Examples of detection systems may be intrusion detection systems (IDS) and intrusion prevention system (IPS). Finally, an example of reaction systems may be anti-virus (AV).

In this context, honeypots add value to each of the three strategies [3]. For what concerns prevention, a honeypot may inhibit an attack because an attacker may be wasting time focusing his resources to exploit a honeypot instead of the actual target. For what concerns the detection, since all connections to the honeypot are probes, scan or attacks, the system can provide valuable information. Finally, for what concerns the reaction, the honeypots may help to study the attacks: indeed, the traffic collected by the honeypot is not mixed with the not malicious one and thus it is possible to develop strategies to counteract that type of threat.

Therefore, the main purpose of the honeypots is to collect information about intrusion attempts, and, in a smaller way, to distract attackers from their actual target. However, it is important to note that honeypots should not be considered as an implementation to solve a specific problem but as a general concept: indeed, the scope of a honeypot changes depending on where and how honeypots are deployed [3].

As reported in [3], the main advantages of honeypots are:

- Valuable Data Collection: since the collected traffic is not mixed with not malicious traffic, the datasets are smaller and thus the analysis is less complex;
- Complete Detection: honeypots capture everything that is sent to them, also unknown strategies and novel pattern attacks.
- Flexibility: honeypots are flexible, they can be used together with specific tools

to react to particular tasks.

Honeypots can be classified according to the following features [20]: level of interaction, deployment environment, resource type, and implementation.

- **Level of interaction**. Depending on the level of interaction, it is possible to consider three categories: low-interaction honeypots (which emulate just a subset of services and do not provide any access to the operating system to the attacker), medium-interaction honeypots (that provide simulated services more elaborated than the previous category and higher interaction for the attacker but no operating system) and high-interaction honeypots (that provide the attackers a real operating system with a large set of services installed).

- **Deployment environment**. There are two general categories of honeypots based on environment: research honeypots (that are generally used by research organisations, are very complex, and provide a large amount of information about attacks) and production honeypots (that are meant to be used in companies, the purpose is to achieve a higher security level and they are easy to use).

- **Resource type**. It is possible to categorize honeypots based on the direction of interaction: server honeypots wait until the attackers initiate the communication, on the other hand, client honeypots actively start an interaction with potential attackers.

- **Implementation**. We can have physical honeypots and virtual honeypots. The former ones employ dedicated hardware, while the latter ones use virtualized machines.

As reported in [21], the number of cybersecurity threats and attacks is growing exponentially and nowadays the standard security measures (access control systems, intrusion detection systems, and firewalls) are no more effective to counteract attackers. Therefore, it is necessary to analyze new threats and develop methodologies to defend ourselves from cyber-attacks. From the first honeypot released in 2000, in the last 20 years, honeypots have had a big impact in the IT security world, becoming very common tools for studying how attackers penetrate security systems. Indeed, the scientific literature is full of honeypot projects but most of all are proof-of-concepts

and out-of-date systems. In 2006 Nawrocki et al. [3] present an extensive overview on honeypots software and methodologies to analyse the related data, highlighting that, on the one hand, the most famous honeypot software are Cowrie[1] (previously Kippo), Dionea[2] and Honeytrap[3], while, on the other hand, the most famous long-terms honeypot projects are the Honeynet Project (1999) and the T-Pot Project, developed from Telekom-Fruhwarnsystem (2012). These two long-term projects present a multi-honeypot platform where low-interaction honeypot tools cooperate intending to be able to reply to more protocols. The honeypots previously mentioned are nowadays active and under maintenance. Tabar et al. [22] in 2020 propose a low-interaction honeypot ecosystem able to emulate IoT devices to understand the attack patterns to IoT systems. Indeed, they emphasize that IoT attacks have an increase of 600% compared to 2016 and thus it is necessary to develop a smarter honeypot able to deal with IoT devices. Since most honeypots are able only to receive passively attacks [3], and they are not able to correctly identify and distinguish the various attack data and scenario, Fan et al. [23] in 2019 propose an efficient honeypot architecture, called HoneyDOC, based on Software Defined Networking (SDN), consisting of three modules, i.e. Decoy, Orchestrator and Captor, to enable all-round design for high-quality attack data capture.

In the following, we propose a brief introduction to the previously mentioned honeypots:

- **Cowrie** is a low-interaction honeypot, the evolution of the no longer supported medium-interaction honeypot Kippo. It can handle SSH and Telnet traffic to observe attacker behaviour. It provides a fake file system, a fake SSH shell and is also able to capture files from the input. Nowadays, as reported in [22], Cowrie is used also to deal with IoT traffic, since many IoT devices still use telnet and SSH protocols.

- **Dionaea**, is a-low interaction honeypot released in 2013, written in Python that supports up to 14 different protocols (HTTP, MYSQL, SMB, MSSQL, FTP, MQTT). It emulates various vulnerable protocols commonly found in a Windows system. The main goal of Diaonea is to capture malware and worms [22].

[1] `https://github.com/cowrie/cowrie`
[2] `https://github.com/DinoTools/dionaea`
[3] `https://github.com/honeytrap/honeytrap`

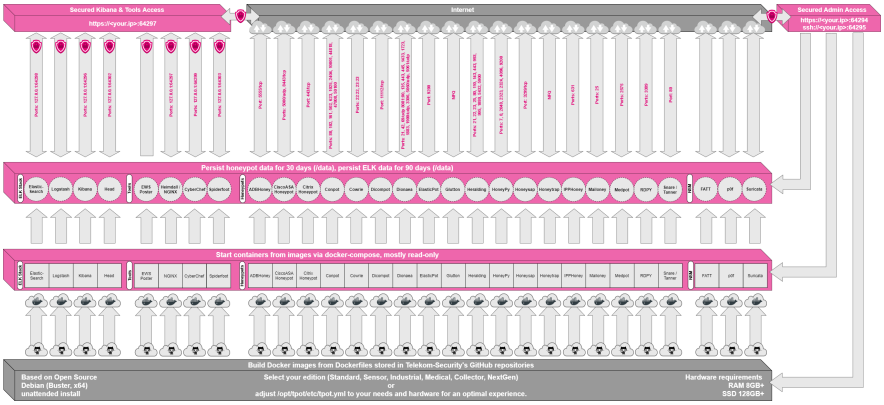


Figure 2.2: T-Pot framework.

- **Honeytrap**, is an open-source framework for running, monitoring and managing honeypots. It can be used to deploy complex honeypot architectures or only to deploy a single server. Furthermore, it is a flexible architecture: it is possible to listen to all ports for detecting threats and collecting information, or listen to a specific port and give predefined answers. Service emulation is not the main focus, only some basic emulation is provided: if for a specific protocol no emulation exists, the default answer is a single character.

- **Honeynet**, according to [24] it is a security-research organization dedicated to learning the black-hat community's tools, tactics, and motives and then sharing any lessons learned. The organization comprises international security professionals who volunteer their time and resources to deploy honeynets that are designed to be attacked. The team then analyzes the information collected from these attacks.

- **Glutton**[4], is the *all eating honeypot*, according to Muhammad Tayyab Sheikh (CS Tayyab), written in GO Lang. It acts as a proxy between attacker and other honeypots and provides the facility to capture, log and analyze the traffic sent between the attacker and the honeypot. It basically listens to all the ports and then acts according to a rule file. The decisions are based on fixed port-based rules.

[4] `https://github.com/mushorg/glutton`

- **T-Pot**[5], is an all-in-one honeypot platform that takes several honeypots and puts them all into a package with some insightful reporting. It includes dockerized versions of 19 honeypots, for example, Cowrie, Dionaea, Glutton and Honeytrap. T-Pot framework gathers all the logs from each honeypot and centralises them into an elastic stack providing the user with a front-end view of all attacks against each service. Samples of malware are also captured providing the ability to further analyse the attacks. It is possible to observe a schematic view of T-Pot in Fig. 2.2. Furthermore, T-Pot provides a simple dashboard interface based on Kibana[6], a data visualization dashboard for Elasticsearch[7] that is a search engine developed in Java providing a distributed search engine with an HTTP web interface.

[5] https://github.security.telekom.com/2015/03/honeypot-tpot-concept.html
[6] https://github.com/elastic/kibana
[7] https://github.com/elastic/elasticsearch

Chapter 3

Comparison of DPI tools

3.1 Introduction

The internet is a continuously growing ecosystem composed of diverse protocols and applications. The rise and spread of smart devices, video-conference platforms as well as the continuous appearance of sophisticated cyber-attacks keeps changing the characteristics of traffic observed in the network. Understanding protocols that are carrying specific flows in the middle of such a variety of traffic has always been essential for multiple applications, in particular for those supporting network security like firewalls and IDS.

Deep Packet Inspection (DPI in short) has been the dominant approach to perform protocol recognition, showing effectiveness in several traffic monitoring scenarios. DPI parses traffic payload searching for signatures that characterize the protocols. Indeed, many DPI solutions do exist and still find important applications, despite the increasing usage of encrypted protocols. DPI is particularly useful in cyber-security scenarios, such as for intrusion detection systems, firewalls and other tools supporting security (e.g., flexible honeypots). The timely identification of a broad range of protocols remains a key first step in the security use case, calling for accurate, efficient and up-to-date DPI solutions. Yet, previous efforts providing an independent evaluation of DPI are already aged [18] or leverage on restricted traffic traces, which questions the applicability of such results to practical scenarios.

In [25] we revisit the question on the quality of DPI-based protocol identification [7, 8, 9, 10]. We select and evaluate four popular, open-source projects imple-

menting DPI, namely nDPI, Libprotoident, Tstat and Zeek. We first study their classification using passively captured traces, covering a wide range of scenarios, i.e., traffic produced by IoT devices, collaborative platforms/video calls, malware, as well as production internet traffic. Establishing a ground truth is challenging when dealing with such diverse traces composed of dozens of protocols. We here evaluate the *consistency* of the classification provided by the tools, relying on heuristics and domain knowledge to validate the decision of each tool when finding conflicting cases.

After that, we investigate whether the DPI solutions operate consistently when exposed to a limited number of packets per flow. Indeed, network applications usually perform protocol identification on-the-fly using the initial packets of each flow, to take timely decisions. For this, we investigate the number of packets per flow each solution needs to reach a decision, as well as the consistency of such decisions as more traffic is observed.

Our results show that:

- All tested solutions perform well when facing traces with well-established protocols. This is particularly true for popular protocols that account for the majority of production traffic;
- Some DPI solutions struggle when facing unusual events, such as massive scans or malware traffic;
- All tested tools reach a final decision already after observing the first packets with payload in a flow;
- nDPI outputs labels more often than others, and it usually agrees with the majority when tools diverge about the protocol of a flow.

To foster further research and contribute to the community, we share our code and the instructions to build the complete datasets exploited in our experiments.[1]

3.2 Datasets and Methodology

Fig. 3.1 summarizes our methodology. We describe the DPI tools selected for testing (Sect. 3.2.1). Then, we build up a set of traces covering different traffic scenarios

[1] `https://smartdata.polito.it/dpi-in-practice/`

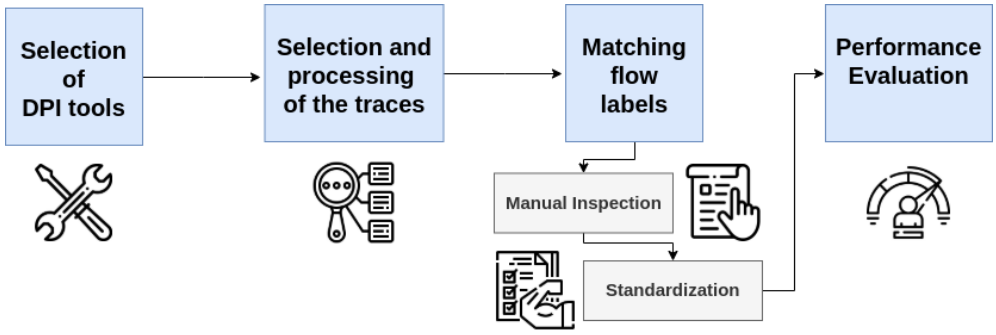


Figure 3.1: Testing methodology.

(Sect. 3.2.2). Next, we process the traces with the DPI tools. As matching the obtained labels requires ingenuity, we perform several steps and build up heuristics to find discrepancies on the final classifications (Sect. 3.2.3).

3.2.1 Selection of DPI Tools

We restrict our analysis to DPI tools that perform *protocol* identification (e.g., HTTP, TLS, SSH etc.), ignoring those aiming at the identification of the services generating traffic (e.g., Google, Facebook etc.) [26, 27]. Namely, we focus on the following four alternatives:

- *nDPI* [9] is an open-source DPI library written in *C* and based on dissectors, i.e., functions that detect the given protocols. It is an *OpenDPI* [28] fork optimized for performance and supports more than 100 protocols.

- *Libprotoident* [8] is a *C++* library that focuses on L7 protocols. It applies a lightweight approach that uses just the first 4 bytes of payload. The idea is to overcome drawbacks of DPI, i.e., computational complexity and privacy risks. The library combines pattern matching with algorithms based on payload sizes, port numbers and IP matching. It supports over 200 protocols.

- *Zeek*[2] – formerly *Bro* [10] – is a complete framework for traffic analysis that also allows L7 protocol recognition. It exploits a combination of protocol fingerprint matching and *protocol analyzers*. It currently supports more than 70 protocols.

[2] `https://zeek.org`

Table 3.1: Flows exported by the different tools before the pre-processing.

Macrotrace	Tool	Flows	
		TCP	UDP
User Traffic	Tstat	681 k	1093 k
	Libprotoident	678 k	1070 k
	nDPI	543 k	1383 k
	Zeek	804 k	1158 k
Media & Games	Tstat	15 k	16 k
	Libprotoident	15 k	14 k
	nDPI	10 k	21 k
	Zeek	17 k	16 k
Malware	Tstat	858 k	979 k
	Libprotoident	858 k	993 k
	nDPI	891 k	1020 k
	Zeek	1242 k	971 k
IoT	Tstat	118 k	50 k
	Libprotoident	118 k	51 k
	nDPI	120 k	62 k
	Zeek	119 k	52 k

- *Tstat* [7] is a passive traffic monitoring tool that classifies traffic flows. It identifies a set of L7 protocols using payload fingerprint matching. It supports over 40 protocols.

Recall that we ignore projects no longer active. In particular, we leave *L7-filter* out since it has been shown to produce unreliable results in more recent scenarios [14]. Equally, we ignore proprietary alternatives, given the intrinsic difficulty to evaluate the root causes of conflicting results without access to source codes [9]. Finally, we do not evaluate *tshark*[3] as it has proofed much slower than the alternatives.

3.2.2 Selection and pre-processing of traces

We consider four scenarios to compare the DPI alternatives, including not only common internet protocols but also protocols encountered by security applications.

We select 421 different PCAP traces that are aggregated in four macro-categories: (i) *User*, which includes ordinary browsing activity of ISP users while at home; (ii) *Media & Games* [29, 30, 31] that includes conference-calls, RTC applications, multimedia and gaming traffic; (iii) *Malware* [32], which aggregates several samples of malware[4] and security experiments;[5] and *IoT* [33, 34], captured in different labs hosting a variety of IoT devices. We include both traces captured in our premises and third-party traces available on public repositories. Traces cover multiple years and total more than 143 GB of PCAP files. For brevity, we do not provide details of each PCAP file here, instead describing only the aggregated *macrotraces*. To allow others to reproduce our results, we link the public PCAP files in our website.[6]

We need to match flows as defined by each DPI tool for comparing their performance.[7] However, tools employ different rules for defining and exporting *flow records*. For example, each tool uses various timeouts to terminate flows that become inactive. Equally, traffic flags (e.g., TCP `FIN` and `RST` flags) are possibly used to identify the end of flows, releasing memory in the traffic monitor. The way such rules are implemented differs and, as a consequence, tools identify and report different numbers of flows. Thus, we need the ingenuity to compare results.

Tab. 3.1 summarizes the number of flows reported by each tool. We see major differences, e.g., Zeek usually identifies more flows than Tstat, even when configured with similar timeouts. This happens because of the way midstream traffic and incomplete flows are processed by the tools.

Most of the cases creating discrepancies are however not interesting for our analysis since they usually refer to flows that carry no payload. Indeed, a lot of flows without payload is present in particular for the Malware traces due to internet scanning traffic. These flows cannot be evaluated with DPI. As such, we perform a *pre-processing step*

[3] `https://www.wireshark.org/docs/man-pages/tshark.html`

[4] `https://www.malware-traffic-analysis.net`

[5] `https://www.netresec.com/?page=PcapFiles`

[6] `https://smartdata.polito.it/dpi-in-practice/`

[7] We use the classic 5-tuple definition for a flow: Source IP address, destination IP address, source port, destination port and transport protocol.

Table 3.2: Macrotraces characteristics with pre-processing results.

Macrotrace	Flows			Packets	
	TCP		UDP		
	Complete	Ignored		Original	Filtered
User	440 k	241 k	1.1 M	118 M	10.1 M
Media&Games	11 k	4 k	16 k	81 M	2 M
Malware	392 k	466 k	979 k	33 M	26 M
IoT	39 k	79 k	50 k	5 M	2 M

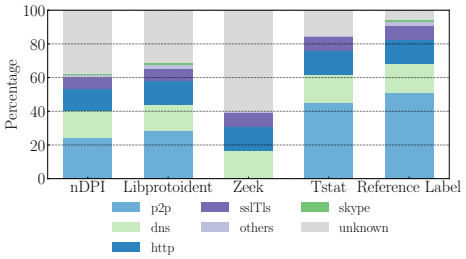
using Tstat as a reference to keep in the final macrotraces only complete flows, i.e., UDP flows with payload and TCP flows with complete three-way handshake. All remaining flows are discarded. Whenever possible, we set the tools with similar timeout parameters for the experiments that will follow. We next normalize results ignoring the small percentage of flows that are not revealed by tools other than Tstat to avoid artefacts related to the way flow are expired or terminated. At last, we keep only the first 20 packets per flow in the final macrotraces to speed up the analysis (see column "Filtered"). We will show later that all tested tools achieve a final protocol classification using a small number of packets per flow. As such, this pre-processing step does not impact results.

We report a summary of the final macrotraces in Table 3.2. We show the number of packets and flows reported by Tstat, with the latter split as TCP and UDP. For TCP flows, we detail the number of *complete* and *ignored* flows.

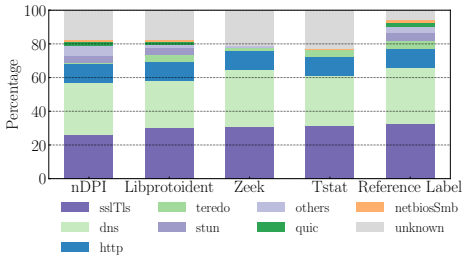
In total, our final macrotraces include more than 3 M flows, and 40 M packets after all pre-processing steps are applied.

3.2.3 Matching flow labels

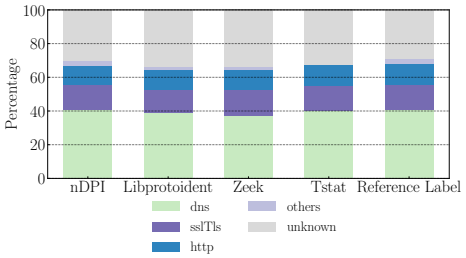
We need some ingenuity to normalize the output of the tools and compare their classifications. First, we normalize all labels, e.g., using always lower case and removing special characters. Then, we manually verify the output strings to identify possible synonyms used across tools. Table 3.3 reports a subset of labels that require manual standardization. In total we manually evaluated 225 labels, replacing cases such as those in the right column of Table 3.3 by a single common label (left column).



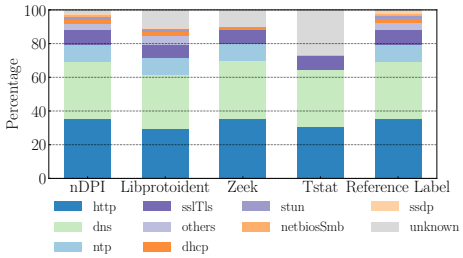
(a) User Traffic Macrotrace



(b) Media & Games Macrotrace



(c) Malware Macrotrace



(d) IoT Macrotrace

Figure 3.2: Percentage of labelled flows for each tool. The last bar in the plots reports percentages for our reference label.

Next, we face the question of how to determine the label for each flow in absence of ground truth. Indeed, the lack of ground truth has pushed most of the previous works to resort to testbeds or emulated traffic that we want to avoid [13]. We thus decide to focus on the *consistency* of different tools, i.e., we assume that the most common normalized label assigned to a flow is the *reference label* for such flow, and calculate a *confidence score* for each decision. In case of conflicts, we manually verify each case.

Table 3.4 reports examples of classification, along with the per-flow *confidence score*. The easiest cases happen when there is a unanimous decision towards the same protocol (e.g., Flow 1) or towards the unknown label (e.g., Flow 2). Both decisions result in a score equals to 1. When at least one tool is able to recognize the protocol, we ignore the unknown labels and pick the recognized label as the reference label. Yet, our *confidence score* is lower in this case, e.g., see Flow 5. It rarely happens (e.g., Flow 6) that all tools recognize a different protocol, or there is a draw (e.g., Flow 7). Some

Table 3.3: Label standardization

Standardized Label	Original Label
p2p	p2p, edonkey, emule, ed2k, cacaoweb, kademlia, bittorrent, torrent
netbiosSmb	netbios, smb, smb2, nbns
krb	krb, kerberos, spnego-krb5spnego
dns	dns, llmnr, mdns
sslTls	ssl, tls
skype	skype, skypetcp
ldap	ldap, cldap
quic	quic, gquic

Table 3.4: Example of flow label constistency and score.

Flow ID	Tool				Reference Label	Score
	Tstat	Libprotoident	nDPI	Zeek		
1	krb	krb	krb	krb	krb	1
2	unk	unk	unk	unk	unk	1
3	krb	unk	krb	krb	krb	0.75
4	unk	unk	krb	krb	krb	0.5
5	unk	unk	unk	krb	krb	0.25
6	unk	sip	unk	p2p	conflict	0
7	krb	krb	p2p	p2p	conflict	0

of these cases have been solved by inspecting the source code of the DPI tools, e.g., giving preference to labels found by pattern matching over those guessed based on port numbers or other heuristics. The few cases we could not resolve are ignored, with a confidence score equals to zero.

Finally, once the reference labels are defined, we calculate performance metrics for each tool. We consider the following metrics: (i) accuracy, the percentage of flows with label matching the reference; (ii) precision (per protocol), the percentage of such flows that match with the reference; and (iii) recall (per protocol), the percentage of

such flows the tool has classified as the given protocol.

3.3 Results

We show a summary of the identified flows per tool and we summarize the classification performance in the several scenarios. Next, we discuss the performance in terms of the number of packets required to reach a steady classification and briefly discuss the computational performance of tools.

3.3.1 Labelled flows per protocol

Fig. 3.2 shows a breakdown of the number of labelled flows reported by each tool. Four plots depict results for the different macrotraces. The last bar on each plot reports the percentage of flows given by our *reference label*, i.e., the label selected by the majority of tools. Each figure reports the most common labels in order of popularity.

In the User Traffic case (top-left plot), Tstat shows the best performance, reporting labels for around 85% of the flows. All the libraries recognize popular protocols (e.g., HTTP, DNS and TLS), but Libprotoident, nDPI and Zeek fail to recognize some P2P traffic, thus leaving a larger number of flows marked as unknown. Yet, notice how the number of unknown flows is small for the reference label – i.e., flows marked as unknown by Tstat are recognized by others.

In the Media & Games case – Fig. 3.2(b) – all tools recognize close to 80% of the flows. This trace is mostly composed of HTTP, DNS and TLS traffic, which are well recognized by all tools. The reference label reports again a lower percentage of unknown than each single tool, showing potential for achieving higher classifications by merging the output of different tools.

The analysis of the Malware macrotrace – Fig. 3.2(c) – leads to worse numbers for all cases. The percentage of labelled flows ranges from 66% to 70%. Here the presence of UDP scans towards multiple ports impact results. Manual inspection shows the presence of payload that matches the fingerprints of scan UDP attacks against certain IoT devices. None of the tools is able to identify the protocol of this malicious traffic, calling for specialized DPI approach in security use cases.

In the IoT case – Fig. 3.2(d) – nDPI is the best performing, labelling almost all

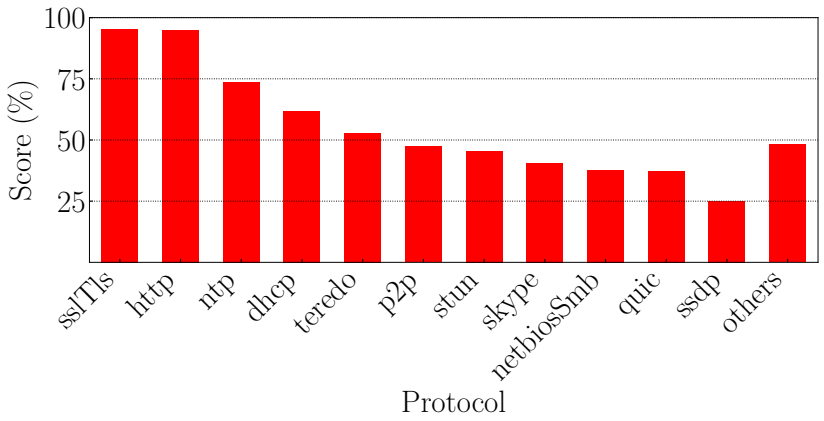


Figure 3.3: Average per flow confidence score for the top reference labels.

flows. Tstat is penalized by the lack of fingerprints for NTP, STUN and SSDP. All in all, most flows in this trace are labelled by at least one tool (see the reference label bar).

Finally, we evaluate the average confidence scores for different protocols. With this analysis, we aim at identifying protocols for which the tools demonstrate high consistency. Fig. 3.3 shows the average scores for flows labelled with one of the top-20 protocols considering all four macrotraces. Common protocols such as TLS, HTTP and NTP are recognized with an average score higher or equal to 75% (left side of the figure). That is, such protocols are consistently identified by at least three tools on average. As we move to less popular labels, the confidence scores reduce significantly. Indeed, the score is reduced to around 25% for Netbios, QUIC and SSDP (right side of the figure). In other words, only one tool outputs a label for flows carrying these protocols, with others marking flows as unknown.

3.3.2 Classification performance

We next quantify the percentage of flows classified by each tool as well as their classification performance in respect to the reference labels. Results are presented in Tab. 3.5. We highlight in bold the best performing tool per trace and metric.

Consider the first-row group in the table. It reports the percentage of labelled flows, summarizing the results presented in the previous section. As said, Tstat reports more

labels for the User Traffic scenario, thanks to its abilities to spot P2P flows. nDPI instead reaches the largest percentages in the other scenarios, thanks to its capabilities to guess labels based on multiple heuristics.

Considering accuracy (second-row group), we see numbers similar to those for labelled flows across all scenarios. That is, the overall accuracy (with regards to the reference labels) is driven by the percentage of unknown flows reported by each tool. Yet, some particular cases can be noticed, such as minor differences between nDPI and Libprotoident in the Media & Games Macrotrace. These minor mismatches arise from cases in which one of the tools, although capable to label the given flow, disagree with the label given by the majority. As we see in the table, these cases are rare and indeed confirm that once tools labels a flow, the provided label is usually reliable.

Zeek wins when it comes to the average precision per protocol (third-row group), almost always reaching 100%. That is, when Zeek recognizes a protocol, its label matches the reference. Yet, Zeek suffers in terms of average recall (fourth-row group), due to its limited set of labels. Libprotoident, on the other hand, reaches the highest average recall per protocol in most scenarios, which can be explained by its large set of labels, with over 200 protocols. nDPI shows balanced numbers for both precision and recall per protocol. nDPI find a good number of labels (high recall) that usually match with the reference (high precision).

3.3.3 How many packets are needed for DPI?

We analyze the performance of tools while limiting the number of packets per flow. This test has been performed by cutting off each flow after observing its n first packets *with payload*, i.e., ignoring initial TCP handshake packets. Flows composed by n or less packets with payload are kept untouched. The goal is to evaluate the number of packets needed to reach a final classification, and whether labels change as more packets are observed.

Fig. 3.4 shows the accuracy for each macrotrace. Clearly, results do not change when increasing the number of packets, and all tools reach an almost steady classification after just one packet. Some tools (e.g., nDPI) increase accuracy further after observing the second packet with payload, but gains are marginal. This result is particularly relevant, as DPI tools are often used for real-time identification of protocols on security applications. Note that nDPI has average accuracy slightly superior to others,

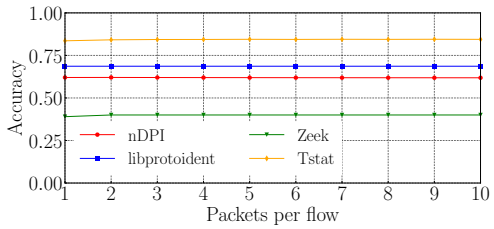
Table 3.5: Summary of classification results.

Metric	Library	Macrotrace			
		User Traffic	Games & Media	Malware	IoT
Labelled Flows	Tstat	**0.85**	0.77	0.67	0.73
	Libprotoident	0.69	**0.86**	0.66	0.89
	nDPI	0.63	**0.86**	**0.70**	**0.98**
	Zeek	0.40	0.78	0.66	0.89
Accuracy	Tstat	**0.85**	0.77	0.67	0.73
	Libprotoident	0.69	**0.82**	0.66	0.85
	nDPI	0.62	0.79	**0.70**	**0.98**
	Zeek	0.40	0.78	0.66	0.89
Average Precision	Tstat	0.99	0.87	0.98	**1**
	Libprotoident	0.96	0.91	0.99	0.80
	nDPI	0.93	0.89	**1**	0.99
	Zeek	**1**	**0.97**	**1**	**1**
Average Recall	Tstat	0.71	0.62	**1**	1
	Libprotoident	**1**	**0.89**	**1**	0.94
	nDPI	0.82	0.78	**1**	**1**
	Zeek	0.66	0.62	0.97	0.79

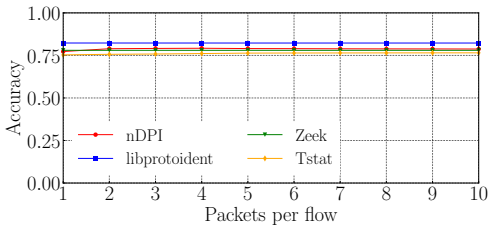
with Libprotoident and Tstat coming next.

3.4 Benchmarking

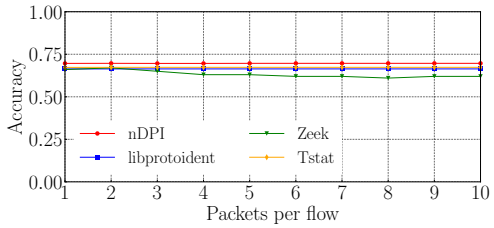
Finally, we also controlled the performance of the tools in terms of memory fingerprint and processing time. Here a general conclusion is hard to be reached since the tools are delivered for different target scenarios. For example, the basic installation of Zeek runs as multiple processes, prepared to handle several Gbps. Libprotoident and nDPI are libraries that can also be integrated into simple demonstration programs. In our tests, all tool, but Zeek, present similar performance figures when processing a single



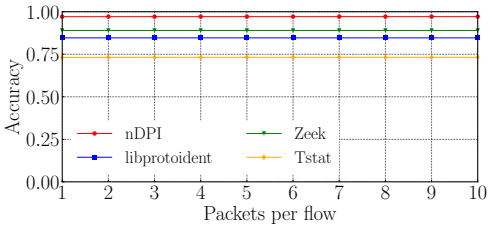
(a) User Traffic Macrotrace



(b) Media & Games Macrotrace



(c) Malware Macrotrace



(d) IoT Macrotrace

Figure 3.4: Accuracy when increasing the number of packets per flow.

PCAP at a time.

We compute the time elapsed from the initial call of the tool until the end of the trace processing. We also measure the peak memory occupation with the *Valgrind* memory profiler. Only results related to the User Traffic macrotrace (Tab. 3.6(a)) and the Malware macrotrace (Tab. 3.6(b)) are shown for brevity. We see that the peak memory occupation varies consistently across the traces and the libraries, Libprotoident and Tstat being the best ones for User and Malware traces, respectively. When it comes to the execution time, Tstat, Libprotoident and nDPI yield comparable results outperforming Zeek by about one order of magnitude.

The time has been calculated from when we called the library until it has finished while the occupation memory has been calculated by using Valgrind[8]. Since all the libraries are set in a shared virtual machine, to provide stable and reproducible results we run all these benchmarking tests at night, 5 times each, and then we calculated the 95% confidence interval. In Tab. 3.6(a) it is possible to observe the results related to a

[8] `https://valgrind.org/info/`

Table 3.6: Peak memory occupation and processing time.

(a) User traffic

Library	Memory (MB)	δ_M	Time (s)	δ_T
Tstat	146.49	0	17.45	0.65
Libprotident	**68.40**	0.04	**9.74**	0.35
nDPI	140.67	0	18.00	0.65
Zeek	992.65	18.49	419.76	8.59

(b) Malware traffic

Library	Memory (MB)	δ_M	Time (s)	δ_T
Tstat	**51.69**	0	16.62	1.57
Libprotoident	56.86	0.12	**6.38**	0.77
nDPI	420.01	0	18.05	1.29
Zeek	273.13	26.98	212.92	6.76

preprocessed user traffic trace (2.6 GB, 4M packets, 286k flows) while in Tab. 3.6(b) the results related to a preprocessed malware traffic trace (2.1 GB, 4M packets, 347k flows). It turns out that Libprotoident reaches the highest performance in term of peak memory occupation and time.

3.5 Outcome

We presented an evaluation of DPI solutions in several traffic scenarios, comparing the consistency of their classifications. The tools are practically equivalent when the input traffic is composed of popular and well-known protocols (e.g., HTTP, DNS and TLS). When applied to complex scenarios, such as to traffic generated by Malware scans, DPI tools struggle. We also observed discrepancies in the classification of less popular protocols, with some protocols being supported by only one of the tools. In sum, there is space for improving these DPI tools by extending their label sets. Interestingly, tools reach steady-state classification after one packet, suggesting they can be exploited in online scenarios. Indeed, for our final purpose, we chose to exploit nDPI since it provides a good trade-off between accuracy and the number of supported protocols.

Chapter 4

Implementation

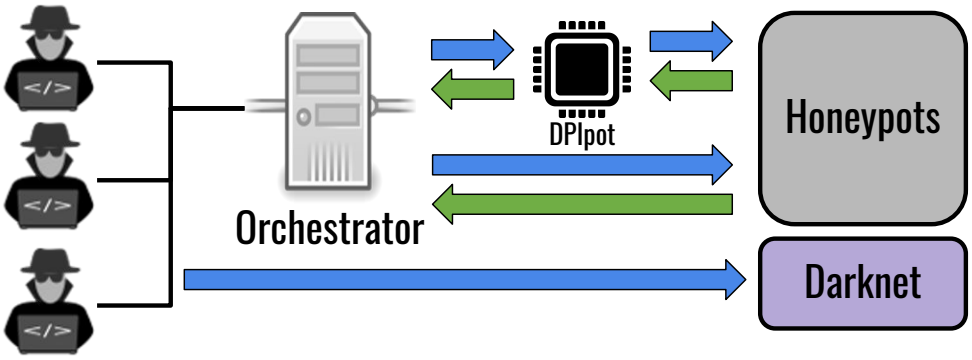


Figure 4.1: Final configuration.

Fig. 4.1 shows the complete honeypot framework which includes DPIpot. Starting from the left we observe the following entities: the attackers, the Orchestrator, the DPIpot, the other honeypot deployments that we take into account and the Darknet infrastructure. The attackers are the remote hosts that try to attack our exposed services: they represent the main subject of our study. The Orchestrator is in charge of coordinating the honeypot system, steering traffic to diverse backends based on flexible rules. The DPIpot is a honeypot that forwards traffic to backend systems after performing DPI classification. The remaining deployments are state-of-the-art honeypots, in our case T-Pot [1] and a Layer 4 Responder [35] that responds to SYN requests, allowing the remote host to complete the three-way handshake, and stores the first packet

[1] `https://github.com/telekom-security/tpotce`

with payload sent by the remote source, if any. Eventually, there is the Darknet, a set of advertised IP addresses that capture remote packets without offering any service or performing any active request. Note that such configuration is flexible and extensible to any honeypot, according to the user's needs: it is necessary to install the honeypot and modify a configuration file. In our case, we decided to rely on T-Pot, since it contains a dockerized version of the most updated state-of-the-art honeypots.

The system must be able to handle different IP addresses and ports on which different honeypots are deployed. To create a flexible and dynamic framework, the idea is to decouple, on one hand, the internal IP address and the port on which a honeypot is deployed in our system and, on the other hand, the IP address and ports that are seen from outside. In this way, it is possible to change, for example, the ports or the set of ports to which a specific honeypot replies.

T-Pot and the other honeypots are deployed in a virtual machine for security purposes: we want that the attackers exploit only our honeypots and not our entire system.

4.1 Orchestrator

The Orchestrator steers the received traffic to back-end systems in a flexible way. It is built on top of iptables rules, a user-space utility program that allows a system administrator to configure the IP packet filter rules of the Linux kernel firewall. The filters are organized in different tables, which contain chains of rules for how to treat network traffic packets.[2] The whole system is configured by means of a *yaml* file. In Listing 4.1 and Listing 4.2 it is possible to see an example of the configuration. In particular, in Listing 4.1 it is possible to specify, for each of the deployed honeypots, the related IP address and port on which we want to deploy them. We here depict three examples, namely the `l4responder`, `tpot-honeypots` and `dpipot`. Note that in the `dpipot` is also present a keyword called `backend`, that lists the available honeypot to which the DPIpot, after having recognized the protocol, can forward the received packet. For now, we are able to correctly reply to 4 protocols: SSH, HTTP, TLS and RDP. By default, if no specific protocol is found, it redirects all the traffic to the L4-Responder.

Instead, in Listing 4.2 it is possible to see an example of the configuration file used to set the iptable rules. For each honeypot, there are: the IP destination, i.e. the

[2] https://linux.die.net/man/8/iptables

destination IP of the incoming packet, the transport protocol and the ports on which the honeypot is listening. Thus, for example, all the packets directed to IP x.x.x.8/29, whatever the port is, are redirected to the `l4responder` that, indeed, listens on ports 0:65535, i.e. all the ports. Instead, if a packet arrives on IP x.x.x.136/29, it is redirected to `dpipot` that chooses the most suitable honeypot to forward the packet according to the results of the nDPI service.

This configuration file makes the framework flexible: it is simple to modify the configuration of existing honeypots or adding new ones.

```
hp:
    l4responder:
      type: "l4responder"
      address: "x.x.x.9"
      port: "1313"
    tpot-hp:
      type: "vm"
      image: "tpot-hp"
      address: "x.x.x.3"
    dpipot:
      type: "dpi"
      address: "x.x.x.9"
      port: "1212"
      backend:
        ssh: "tpot-hp:22"
        http: "tpot-hp:80"
        tls: "tpot-hp:443"
        rdp: "tpot-hp:3389"
        default:"l4responder"
```

Listing 4.1: Orchestrator code: example of honeypots.

```
iptables_rules:
# reply all
    - honeypot: "l4responder"
      interface: "bond0"
      ip_dst: "x.x.x.8/29"
      proto: "tcp"
      ports: "0:65535"
    - honeypot: "tpot-hp"
      interface: "bond0"
      ip_dst: "x.x.x.72/29"
      proto: "tcp"
      ports: "0:65535"
    - honeypot: "dpipot"
      interface: "bond0"
      ip_dst: "x.x.x.136/29"
      proto: "tcp"
      ports: "0:65535"
```

Listing 4.2: Orchestrator code: example of iptables configuration.

4.2 DPIpot

DPIpot is a honeypot that forwards the received packets to the correct honeypot, after performing DPI classification.

From an implementation point of view, the code has been developed in Python, with the following specific libraries: Thread, to exploit concurrent programming,

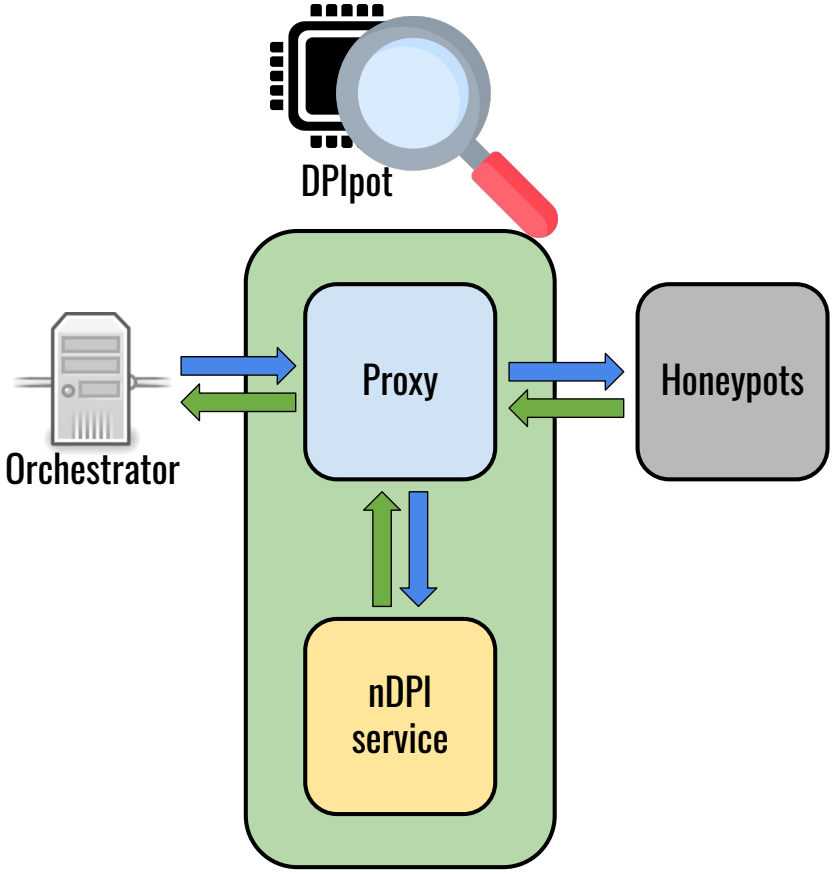


Figure 4.2: DPIpot overview.

Socket, to create virtual communication channels between nodes, and Scapy, to create customized packets.

The tool used for DPI is nDPI, as a result of the DPI analysis presented in Chap. 3. nDPI is used as an independent service, as we observe in Fig. 4.2: for each new flow, the first packet is sent through to the nDPI service. nDPI performs deep packet inspection in real-time and outputs the protocol label. This label is compared against the configuration file: if a suited honeypot exists for that label, a new socket towards that honeypot is opened and all the packets belonging to that flow are redirected to it.

In the following, we present a brief scheme of the implementation.

1. For each honeypot present in the yaml configuration file in the section `backend`, we instantiate a `DPIProxy` thread object (the light green block in Fig. 4.2). Every

block waits for incoming connections.

2. When a new connection arrives, a new element in the flow table is created. Simultaneously, also a socket is created. Three fake packets are created and sent to the nDPI service since it always needs a three-way handshake to work correctly. Then, when the first packet with payload arrives, it is sent to the nDPI service to be analyzed.

3. When the nDPI service returns the label of the packet, if a suited honeypot exists, the packet is forwarded to such backend.

The flow table is a crucial element of this infrastructure, and for this reason, it has to be efficient: since it can grow exponentially, we have to make sure that its size remains limited during the time. Therefore, we develop a specific function that, after a customized time, checks for expired flows and cleans up the table.

4.3 Benchmarking

In this Section we show the performance of the DPIpot. We show the number of active connections that the honeypot is able to simultaneously handle. We used Siege[3], an open-source regression test and benchmark utility for the HTTP protocol.

Fig. 4.3 shows how many concurrent connections DPIpot can handle compared to an Apache webserver. To perform this test, we installed DPIpot in a closed environment, a virtual machine. In each run of the test, which lasts 1 minute, an increasing number of concurrent users, from 1 to 150, tries to open the maximum number of connections. The y-axis reports the rate, i.e. how many successful connections are correctly opened in the unit of time. We observe that the performances of the DPIpot are not so good compared to an Apache server, with a difference of three orders of magnitude. This poor result is because the code, written in Python, is not yet optimized to handle lots of contemporary users. This version of the DPIpot must be considered as a preliminary version that will be further optimized in the future.

[3]https://github.com/JoeDog/siege

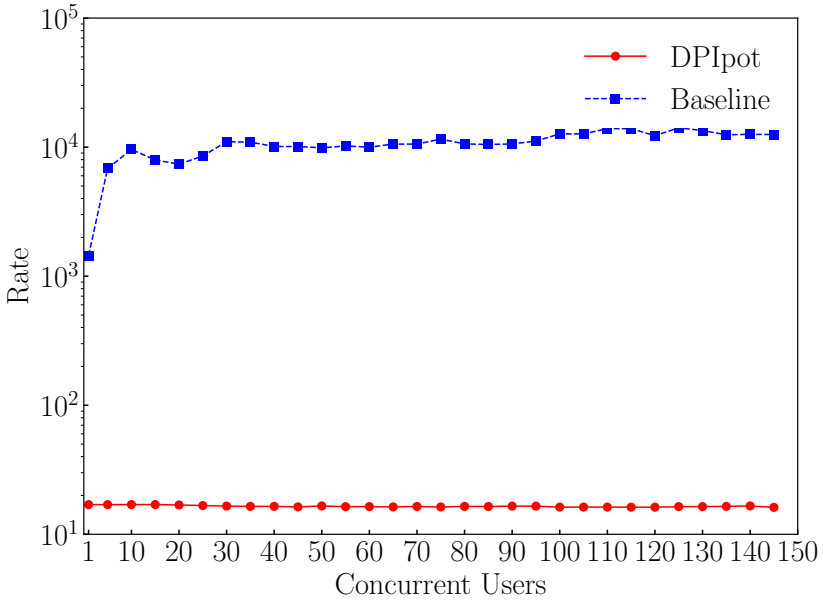


Figure 4.3: DPIpot transaction rate.

Chapter 5

Analysis of the captured traffic

In this Chapter we propose an analysis of the traffic captured in our final deployment, intending to understand the impact of different deployments on the type of information obtained, and whether these possible variations impact the type of information obtained by the honeypots. In Section 5.1 we present our methodology: in Section 5.1.1 we define the considered scenarios, in Section 5.1.2 we show the characteristics of the analyzed traffic, in Sections 5.2, 5.3, 5.4 and 5.5 we present the final results. The fundamental aim of the Chapter is to reply to the following questions:

- What is the share of the traffic reaching the honeypots that arrives at different attack phases (i.e., just probing open ports, establishing the three-way handshake, actually sending packets or performing login attempts)? Which is the percentage of traffic not opening a connection?
- Does the attack pattern change if we start replying to all the connection requests?
- Does the attack pattern change depending on the kind of services we expose?
- Does identifying protocols on-the-fly before replying, even when traffic reaches non-standard ports, influence the attack patterns?

5.1 Datasets and Methodology

Fig. 5.1 summarizes our methodology. We describe the different scenarios that represent different deployments. Then, we process the captured traffic and we characterize

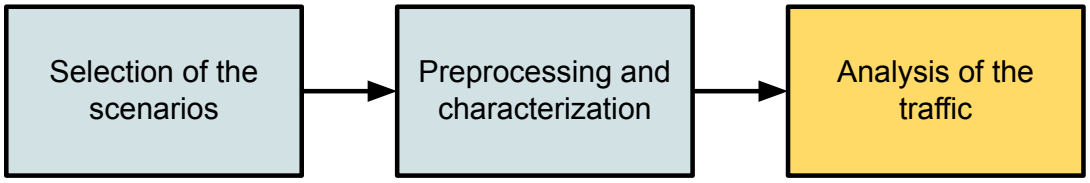


Figure 5.1: Methodology.

the traffic. Finally, we evaluate the impact of the different deployments in terms of volume, sources and services requested.

5.1.1 Selection of the scenarios

	IP	Service	Ports
L4-Responder	130.192.167.8/29 130.192.167.16/29 130.192.167.24/29 130.192.167.32/29 130.192.167.40/29 130.192.167.48/29 130.192.167.56/29 130.192.167.64/29	reply all ports web server, ports proxy (generic and squid) dbserver (mysql, mssql, postgres, mongodb) remote desktop (ms rd, vnc, teamviewer, anydesk) fileserver (netbios, CIFS) + UDP terminal (ssh, telnet) mail server (pop, imap, smtp)	0:65535 80,443 8080,8000,3128 3306,33060,1433,4022,135,1434,5432,27017 3389,5900,5901,5800,5801,5938,6568 135:139,445 22,2222,23,2323 25,110,143,465,993,995
L7-Responder	130.192.167.72/29 130.192.167.80/29 130.192.167.88/29 130.192.167.96/29 130.192.167.104/29 130.192.167.112/29 130.192.167.120/29 130.192.167.128/29	reply all ports web server, ports proxy (generic and squid) dbserver (mysql, mssql, postgres, mongodb) remote desktop (ms rd, vnc, teamviewer, anydesk) fileserver (netbios, CIFS) + UDP terminal (ssh, telnet) mail server (pop, imap, smtp)	0:65535 80,443 8080,8000,3128 3306,33060,1433,4022,135,1434,5432,27017 3389,5900,5901,5800,5801,5938,6568 135:139,445 22,2222,23,2323 25,110,143,465,993,995
DPIpot	130.192.167.136/29	reply all ports	0:65535
Darknet	130.192.166.0/24		

Figure 5.2: Final deployment.

Fig. 5.2 summarizes the different deployments of honeypots. In particular, we consider four infrastructures: the L4-Responder, a level four honeypot able to complete only the TCP three-way handshake; L7-Responder, a framework that gathers the vast majority of nowadays state of the art honeypot; DPIpot, a honeypot that is able to identify on-the-fly the correct protocol and reply as requested by the attacker; and the Darknet composed by 253 IP addresses. From now on we will call T-Pot as L7-Responder

since we want to highlight the difference between replying only at the transport layer or at the application layer. L4-Responder and L7-Responder are deployed on 64 IPs each, in subgroups of 8 IPs depending on the services to which the honeypot is exposed to. Indeed, we select 8 service scenarios according to the following scheme:

- reply to all ports;
- reply to Web Server services;
- reply to Proxy services;
- reply to DB Server services (e.g. MySQL, MSSQL, PostgreSQL, MongoDB);
- reply to Remote Desktop services (e.g. VNC, TeamViewer);
- reply to Fileserver services (e.g. NetBIOS, CIFS);
- reply to Terminal services (e.g. SSH, Telnet);
- reply to Mail Server services (e.g. POP, IMAP, SMTP).

For what concerns the DPIpot, it is deployed on 8 IPs and replies to all ports. Indeed, in Fig. 5.2, in the column IP, it is possible to observe the same structure, and in the column Ports, there are the ports related to the services. For example, the L4-Responder that replies to all the ports (as it is possible to see in the last column Ports) is deployed on 8 IP address, from 130.192.167.8 to 130.192.167.15, L7-Responder that replies to all ports is deployed on 8 IPs from 130.192.167.72 to 130.192.167.89, and the DPIpot is deployed on 8 IPs from 130.192.167.136 to 130.192.167.144.

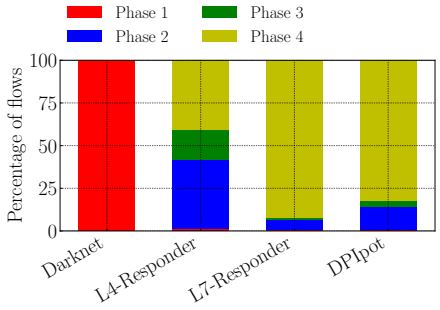
5.1.2 Preprocessing and characterization

In this preliminary phase, we captured 10 days of traffic hitting our infrastructures. To extract all the statistics, all the traffic has been processed with Tstat [11]. We extract the flow table, filter the TCP traffic and we associate a label with the name of the infrastructure to each flow. In total, we consider 62 M flows, 47 M of which hitting the honeypots and 15 M hitting the Darknet. In Tab. 5.1 we observe in detail how the flows are reaching the different infrastructures. The column Complete Flows indicates the numbers of flows that complete the three-way handshake: note that this number

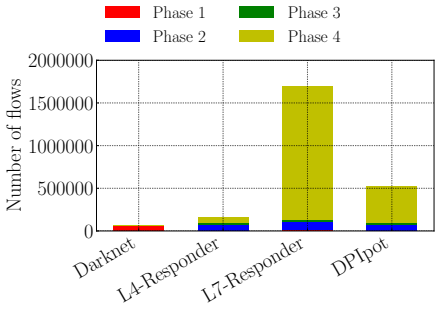
Table 5.1: Characterization of the infrastructure

Infrastructure	**Ports**	**Flows**	**Complete Flows**	**Packets Received**	**Packets Sent**	**IPs**
Darknet	All	15 M	0	16 M	0	253
L4-Responder	All	1 M	765 k	4 M	3 M	8
	Web	513 k	10 k	589 k	46 k	8
	Proxy	573 k	6 k	633 k	27 k	8
	Databases	526 k	30 k	666 k	117 k	8
	Remote Desktop	642 k	124 k	1 M	279 k	8
	Fileserver	587 k	128 k	1 M	476 k	8
	Terminal	615 k	65 k	1 M	419 k	8
	Mail	508 k	9 k	574 k	30 k	8
L7-Responder	All	13 M	13 M	109 M	107 M	8
	Web	508 k	9 k	628 k	110 k	8
	Proxy	568 k	6 k	645 k	35 k	8
	Databases	521 k	24 k	924 k	342 k	8
	Remote Desktop	2 M	1 M	5 M	3 M	8
	Fileserver	10 M	10 M	81 M	83 M	8
	Terminal	1 M	672 k	14 M	11 M	8
	Mail	497 k	4 k	565 k	32 k	8
DPIpot	All	4 M	4 M	16 M	12 M	8

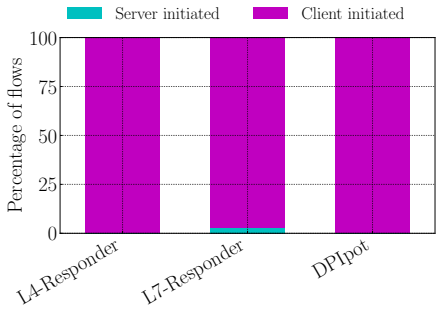
is significantly smaller than the total number of flows. Indeed, when we consider a subset of services, we reply only on some ports; for example, when we consider Web Services, we reply only to port 80 and 443, all the other ports behave like a Darknet: they receive packets but they do not reply. For this reason, the number of complete flows is only those that hit the active ports and thus this number is smaller than all the flows that hit all the ports. The column IPs indicates the number of destination IPs used for each infrastructure.



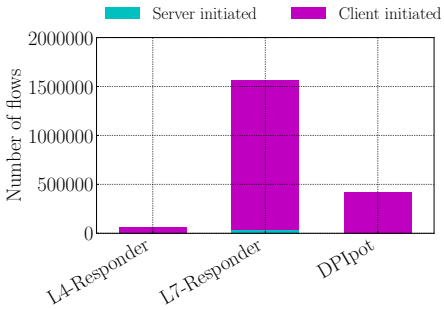
(a) Phases percentage.



(b) Phases absolute number.



(c) Percentage of client-initiated or server-initiated flows.



(d) Absolute number of client-initiated or server-initiated flows.

Figure 5.3: Most contacted ports in the time.

5.2 Different attack phases

In this Section, we want to understand what is the share of the traffic reaching the honeypots that arrives to different attack phases. We define the phases as follows:

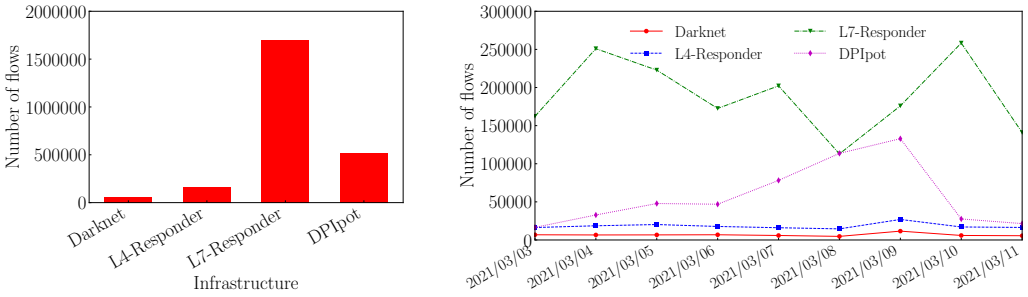
- **Phase 1**: only SYN;
- **Phase 2**: incomplete three-way handshake [SYN + SYN/ACK only];
- **Phase 3**: complete three-way handshake without payload;
- **Phase 4**: complete three-way handshake complete with payload.

Fig. 5.3(a) shows the percentage of flows for each phase, while Fig. 5.3(b) reports the same information in absolute number. As expected, all the Darknet traffic is in phase 1, since, by definition, it does not reply to any packet. For what concerns the L4-Responder, 40% of the traffic is in phase 2 probably due to port scanning (i.e. the attacker only wants to know whether the targeted port is open or not), 20% of the traffic is in phase 3 and 40% of the traffic in phase 4. The traffic in phase 3 is probably caused by server-initiated services, i.e. services that are waiting that after the three-way handshake the server sends the first packet with payload. The traffic in phase 1 is probably due to some infrastructure failure. If we look at L7-Responder, we can observe that the majority of the flows are in phase 4: this is behaviour is because L7-Responder is an enhanced system that handles different honeypots and thus it is able to correctly interact with different types of services. Also in this case, the traffic in phase 1 is probably due to some infrastructure failure. However, it is important to note that the absolute number of flows in phase 2 and 3 are similar to the L4-Responder and DPIpot infrastructures, as shown in Fig. 5.3(b): it can be considered a common behaviour and it may be caused by server-initiated services. For what concerns DPIpot, we observe that the majority of traffic is in phase 4. This means that if we reply, there is an increase in the number of flows with payload. Indeed, in Fig.5.3(b) it is possible to see that there is a significant increase in the absolute number of flows compared to the Darknet: the more accurately we reply, the more traffic we observe.

Furthermore, we differentiate the phase 4 flows into 2 categories: server-initiated and client-initiated. As explained before, server-initiated flows are those that are waiting, after the three-way handshake, for the first packet from the server, while client-initiated flows are those that send the first packet to the server. In Fig. 5.3(c) and 5.3(d), it is possible to observe this division for each infrastructure. As expected, the L4-Responder is only client-initiated since it completes only the three-way handshake. For what concerns L7-Responder, the majority of the flows are client-initiated, except a 5%. In this little percentage, we found out the VNC service, which is often used for exploits. L7-Responder, when a packet arrives on port 5900, the port related to VNC service, uses Heralding[1], a honeypot that is able to propose a credentials user interface and to create password logs.

[1] https://github.com/johnnykv/heralding

5.3 Replying to all connections



(a) Number of flows for each infrastructure.

(b) Time series.

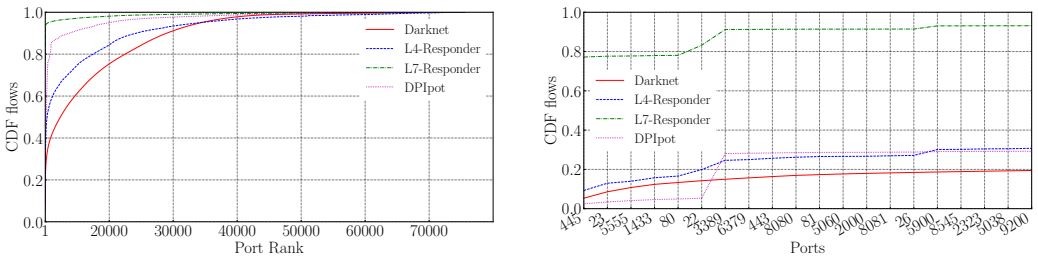
Figure 5.4: General characterization

In this Section, we want to understand if the attack patterns change when we start replying to all connections. In this scenario we consider the traffic hitting the infrastructures that reply to all port, as it is shown in Fig. 5.2, and we use the one hitting the Darknet as a baseline for our comparison. Fig. 5.4(a) shows the number of flows for each infrastructure. We see that, depending on how accurately we reply, the absolute number of flows increases. In Fig. 5.4(b) we show a time series for 10 days of captures. It is possible to see a clear increase in volume for DPIpot during the first 8 days. However, the time interval is still too small to observe consolidated attack patterns in time.

In the following, we analyze the destination ports and the source Autonomous System (AS) to understand if it is possible to discover common attack patterns.

5.3.1 Destination port analysis

In Fig. 5.5(a) we show the CDF of the flows sorted according to the most contacted port of the Darknet. Indeed, the Darknet red curve increases more slowly than the other curves: this means that the distribution of traffic hitting the ports is smoothed. The L4-Responder blue curve increases faster than the red one: some ports attract more traffic. The DPIpot purple curve and L7-Responder green curve increase very fast: they reach



(a) All ports

(b) Only the first 20 ports.

Figure 5.5: CDF of flows per ports in the different infrastructures.

more than 80% of the traffic with the first 100 ports. In particular, if we consider the detailed view in Fig. 5.5(b), L7-Responder reaches 80% of the traffic with only one port, the 445. DPIpot has a jump on port 3389 (RDP service) since we can correctly reply to that service. In Tab. 5.2 we show the mapping between ports and services for the 20 most contacted ports of the Darknet.

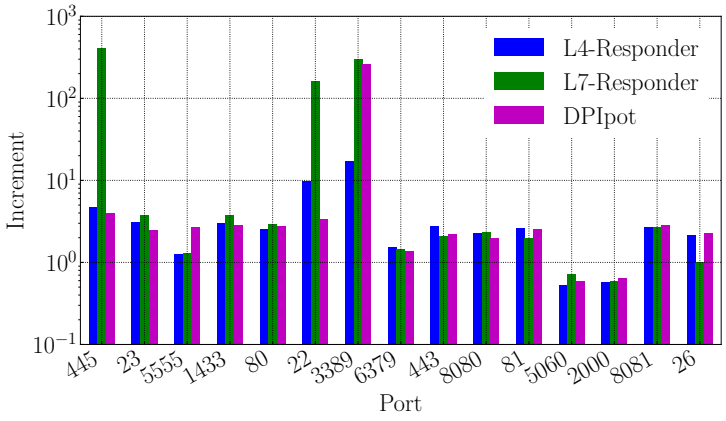


Figure 5.6: Increment with respect to the Darknet per port.

Fig. 5.6 reports a detailed view of the traffic increase on the top-15 ports compared to the Darknet. In particular, we observe a significant increase of traffic in port 445, 22 and 3389. For example, for these services, L7-Responder has an increase greater than 100 times the Darknet. Also DPIpot registers a comparable increase for the RDP

Table 5.2: Ports mapped to services.

Port	**Service**
445	SMB
23	Telnet
5555	VPN
1433	SQL
80	HTTP
22	SSH
3389	RDP
6379	Redis
443	HTTPS
8080	HTTP
81	HTTP
5060	SIP
2000	Callbook
8081	HTTP
26	FTP
5900	VNC
8545	RPC
2323	Telnet
5038	-
9200	WSP

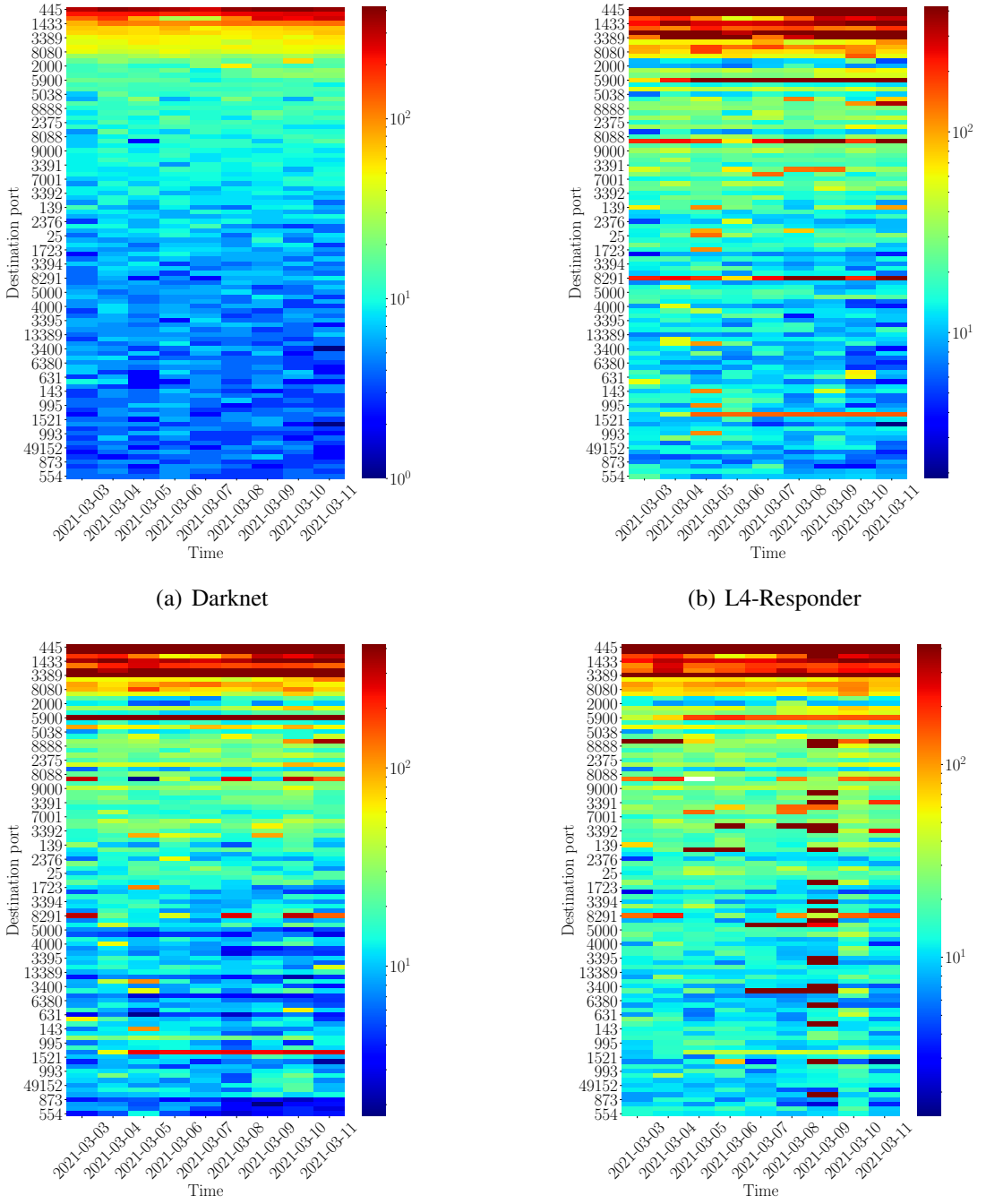
service.

The heatmap in Fig. 5.7 shows the volume of traffic reaching the top-100 port for each of the considered time bins. Note that the ports are ordered according to the way they reach the Darknet and the scale is logarithmic. Fig. 5.7(b) shows that if we start replying at the transport layer, we obtain more attacks: for example, port 5900, 8291, 8728 (after 8088 in the figure) and 5901 (before 1521). It is possible to observe that there are some vertical pattern attacks, for example on day 2021/03/05 there is a clear attack pattern on more than 20 ports. It is possible to observe also some horizontal pattern, for example, port 8291 is contacted each day. Fig. 5.7(c) shows that if we start replying also at the application layer, we obtain a similar behaviour. Fig. 5.7(d) shows that if we are able to identify the protocol on-the-fly, we observe more vertical attack patterns compared to the other cases.

Since we want to quantify the increment of traffic with respect to the Darknet, in Fig. 5.8 it is represented the daily increase of traffic per port, defined as follows:

$$\delta = \frac{heatmap_x - heatmap_{Darknet}}{heatmap_x + heatmap_{Darknet}} \tag{5.1}$$

where $heatmap_x$ represents the matrix associated with the heatmap of each infrastructure, while $heatmap_{Darknet}$ represents the one associated with the Darknet. Thus, δ is bounded between 1 and -1: when the traffic of a honeypot infrastructure is equal to the Darknet, δ is 0, when it is greater, $\delta > 0$ and the graphical representation colour is red, and when it is less, $\delta < 0$ and the graphical representation colour is blue. By analyzing the heatmaps, it turns out that there is a clear increase in the traffic in almost all the 100 ports. In particular, we register an increase in volume on all our infrastructures. Furthermore, we observe that if we start replying, the traffic increase also on other ports: in Fig. 5.8(a), it is clear that also other ports are contacted more than the Darknet, for example, port 5900, 8291, 8728 (after 8088 in the figure) and 5901 (before 1521). For what concerns L7-Responder, the pattern is similar to L4-Responder, although for the most contacted ports the increase is greater and for the previously mentioned ports the pattern is not continuous among days. For example, ports 8728 (after 8088) and 8291 has traffic peaks on different days. The traffic pattern of DPIpot is slightly different: it turns out that if we start reply with the most accurate protocol we can reply with famous protocol also on non-standard ports. For example, there is a peak on port 7777 (the one after 3400) due to RDP requests and on port 9000 due to TLS requests.



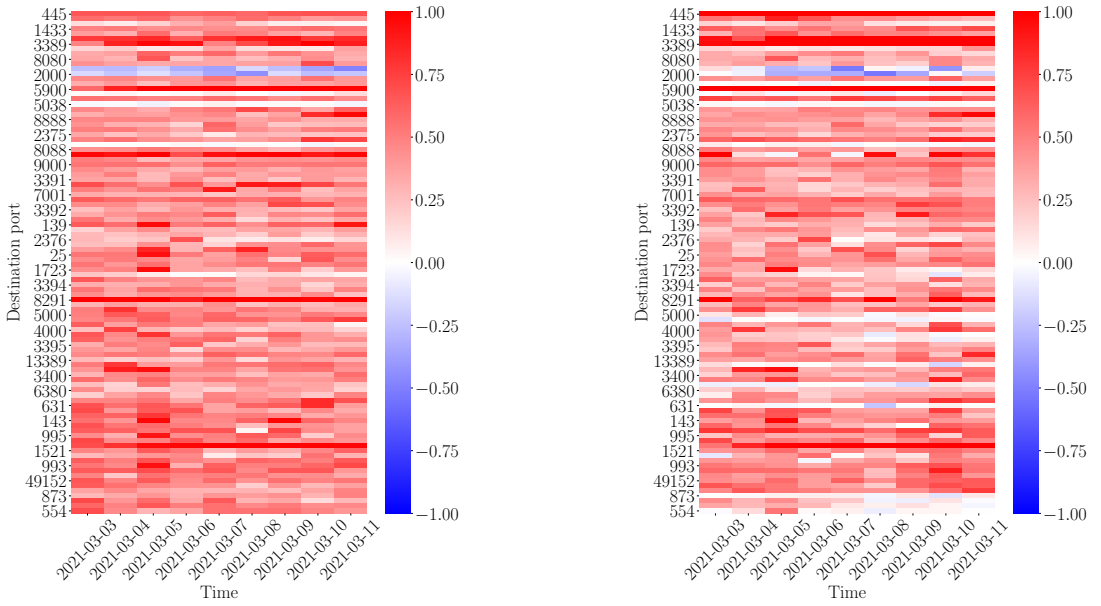
(a) Darknet

(b) L4-Responder

(c) L7-Responder

(d) DPIpot

Figure 5.7: Most contacted ports in the time.



(a) L4-Responder

(b) L7-Responder

(c) DPIpot

Figure 5.8: Delta.

5.3.2 Source Autonomous System analysis

In this Section, we analyze the source AS.

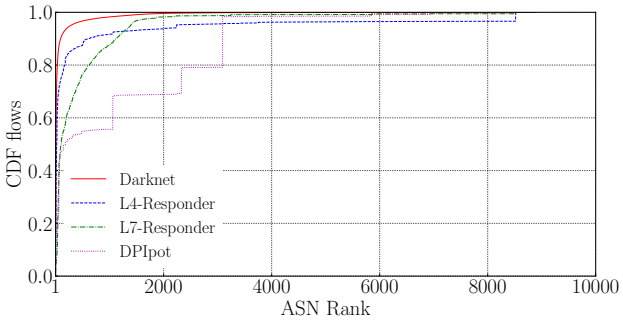


Figure 5.9: CDF of flows per AS in the different infrastructures.

As it is possible to see in Fig. 5.9, most of the Darknet traffic comes from a relatively small number of ASes: the first 30 ASes of the Darknet produces 80% of the traffic. The curves related to L4-Responder, L7-Responder and DPIpot confirm, instead, that the more we reply, the more sources we attract. In some cases, as in the DPIpot one (purple curve), a few sources can produce a relevant amount of traffic: in the figure, the first jump is due to AS59793, from Russia, the second jump is caused AS12334, from Spain, and the third jump is due to AS262150, from Argentina.

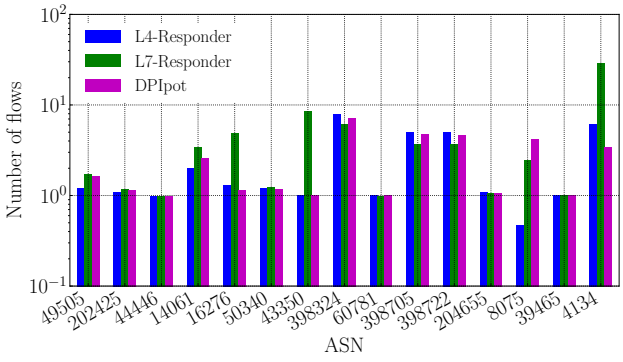


Figure 5.10: Increment with respect to the Darknet per AS.

In Fig. 5.10 it is possible to observe the increment of traffic coming from different

ASes, sorted according the most active ASes of the Darknet. It turns out that there is a clear increase in the traffic for almost all of the 15 Darknet most active AS.

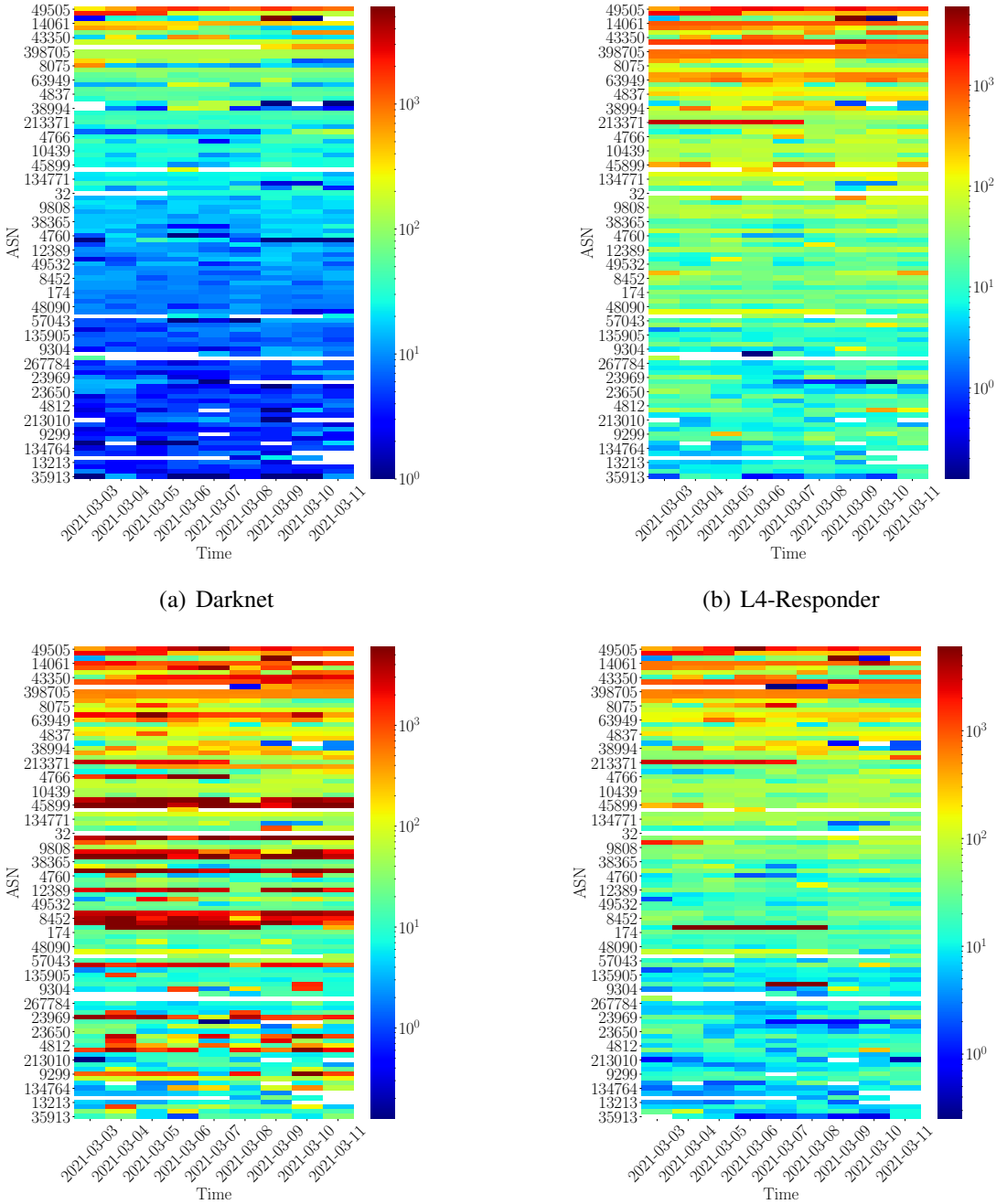
In Fig. 5.11, we analyze the 100 most active ASes in time: also from these heatmaps, it is possible to state that when we start replying, we receive more attacks from different sources. For example, Fig. 5.11(a) shows the heatmap of the Darknet on a logarithmic scale. If we look at the heatmaps of the other infrastructures, we can observe *much less blue*: this means that we are receiving much more traffic than the baseline. For example, Fig. 5.11(c) shows that if we are able to correctly reply to attackers, there is a clear increment of traffic coming from different locations.

In Fig. 5.12 we want to quantify this increasing. As before, we define:

$$\delta = \frac{heatmap_x - heatmap_{Darknet}}{heatmap_x + heatmap_{Darknet}} \tag{5.2}$$

where $heatmap_x$ represents the matrix associated with the heatmap of each infrastructure, while $heatmap_{Darknet}$ represents the one associated with the Darknet. We can note that the intensity of the red colour in Fig. 5.12(b), L7-Responder, and Fig. 5.12(c), DPIpot, is much darker than the L4-Responder in Fig. 5.12(a), especially for L7-Responder. For sure we observe an increase in the L4-Responder, despite it completes only the three-way handshake. For what concerns DPIpot, we observe an increase only for some ASes: the reason is that while with L7-Responder we cover a large set of protocols that we reply, with DPIpot, we reply accurately only with 5 protocols. However, this result is heartening: if we increase the supported protocols of DPIpot, we expect a behaviour similar to L7-Responder.

In conclusion, we affirm that if we start to reply, we observe a huge increment of the traffic both when observing the destination ports and in the source AS. Moreover, when we start replying we observe more attacks not only on the most contacted ports of the Darknet but also on adjacent ports. Even the number of sources and their location becomes wider. In general, we speculate that the increase of traffic depends on how accurately we reply: we note an increasing pattern between L4-Responder, L7-Responder and DPIpot. In particular, with DPIpot, we observe an increase of attacks on non-standard ports with services that DPIpot is able to handle.



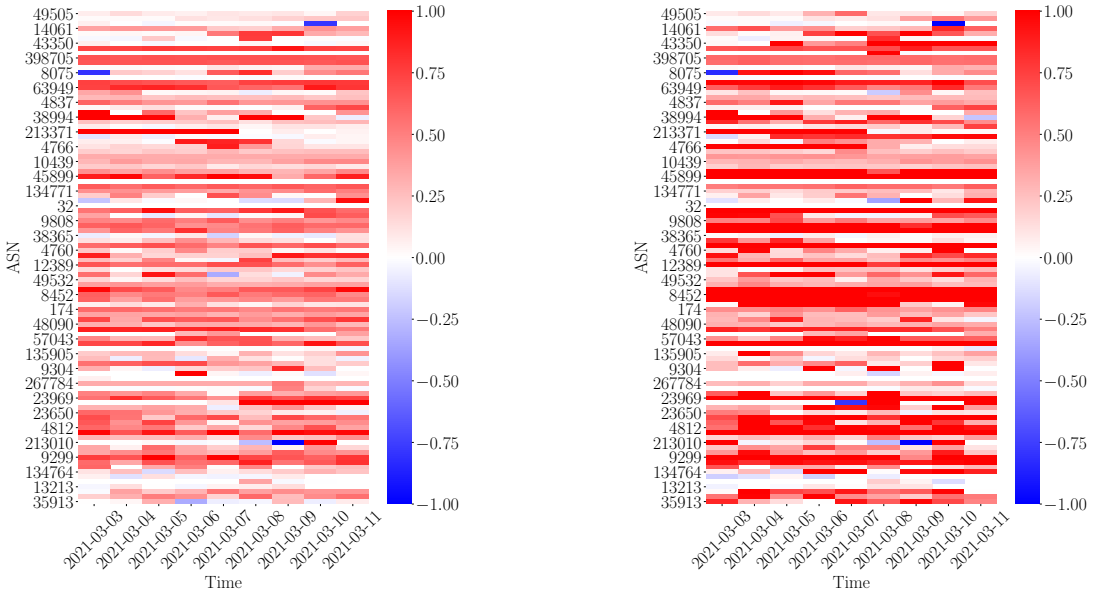
(a) Darknet

(b) L4-Responder

(c) L7-Responder

(d) DPIpot

Figure 5.11: Most contacted ASes in the time.



(a) L4-Responder

(b) L7-Responder

(c) DPIpot

Figure 5.12: Delta.

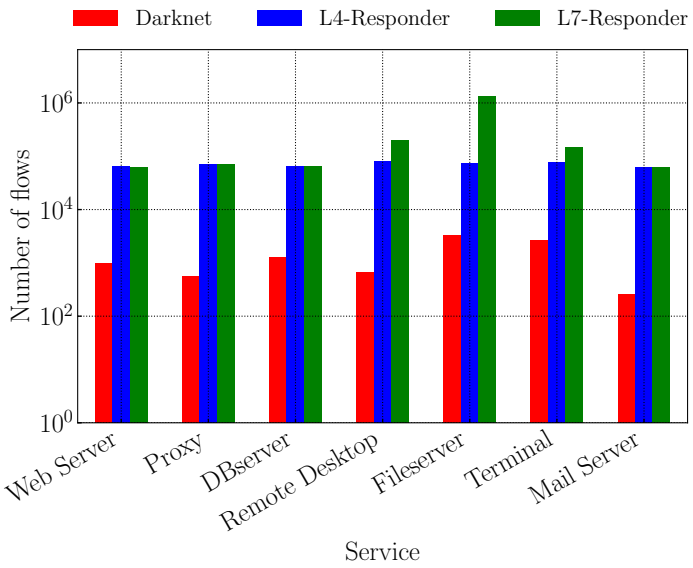


Figure 5.13: Total volume per service.

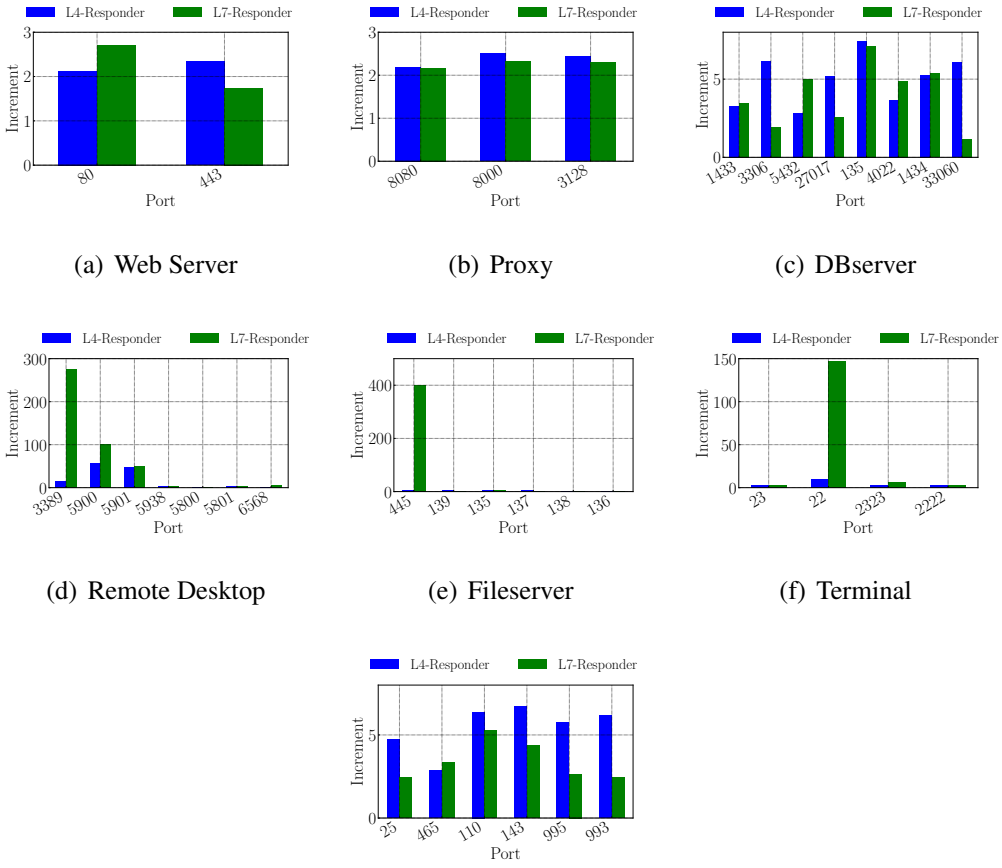
5.4 Exposing different services

In this Section, we want to understand if the attack pattern changes depending on the kind of services we expose. As explained before, we choose to consider 7 groups of services: Web Server, Proxy, DB Server, Remote Desktop, Fileserver, Terminal and Mail Server. We deployed them on 8 IP addresses each, as it is shown in Fig. 5.2.

Fig. 5.13 shows the absolute number of flows per service for each infrastructure. We register a significant increment either with L4-Responder, either with L7-Responder. If we reply with the L4-Responder the absolute number of flows is comparable among the different services. Instead, if we reply with L7-Responder, we observe a larger increase for the Remote Desktop, Fileserver and Terminal services.

5.4.1 Destination port analysis

In Fig. 5.14 we observe the increments compared to the Darknet: in all the seven different configurations the traffic is at least doubled for each service. Furthermore, we observe that L7-Responder present a significant increase for port 445 that is 400 times



(a) Web Server (b) Proxy (c) DBserver

(d) Remote Desktop (e) Fileserver (f) Terminal

(g) Mail Server

Figure 5.14: Increment of the number of flows with respect to the Darknet per service.

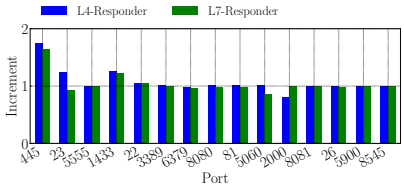
more than the Darknet and 3389 which is almost 300 times more than the Darknet, and port 22 that is around 150 times more than the Darknet. It is interesting to note that in some cases, for example for Proxy services, the amount of flows of the L4-Responder is greater than the one of L7-Responder. The reason may be that since the L4-Responder completes only the three-way handshake, attackers try repeatedly to contact that service. Instead, when we reply with L7-Responder, we are able to satisfy a lower number of requests at a time.

In Fig. 5.15, it is possible to observe the increments compared to the Darknet, of the non-active ports for each service. It turns out that there is not a significant increase in traffic if we reply only to some services. There are some exceptions, for example, port 1433 and 445 receive more traffic also when they are not active. Furthermore, it is interesting to note that there is no significant increase in the traffic if we reply with a layer seven honeypot, L7-Responder, instead of the L4-Responder.

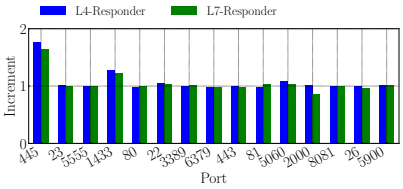
5.4.2 Source Autonomous System analysis

For what concerns the source ASes, in Fig. 5.16 it is possible to observe the increment of traffic with respect to the Darknet. Each plot represents the most active AS that contact a specific group of services. As expected, the increment related to AS is coherent with the increment related to the destination port. Indeed, the increment of traffic-related to Fileserver services, Fig. 5.16(e), that is around 500 times, is consistent with Fig. 5.14(e), in which the increment is around 400.

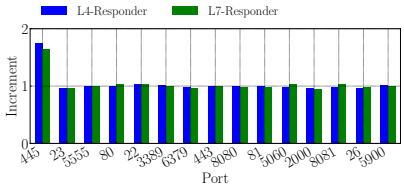
In conclusion, it turns out that when we reply with different services we observe a significant increase of traffic only on the ports with an active service.



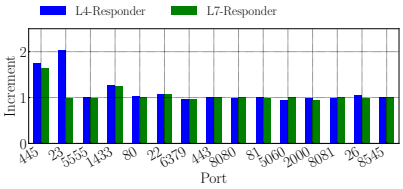
(a) Web Server



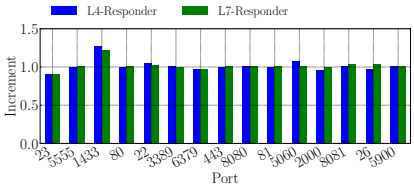
(b) Proxy



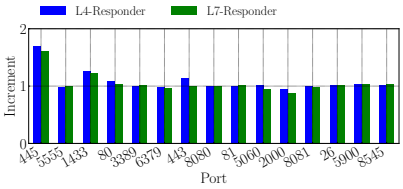
(c) DBserver



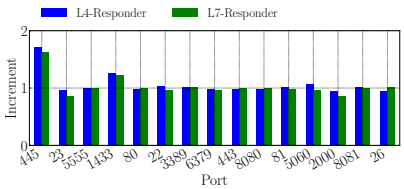
(d) Remote Desktop



(e) Fileserver

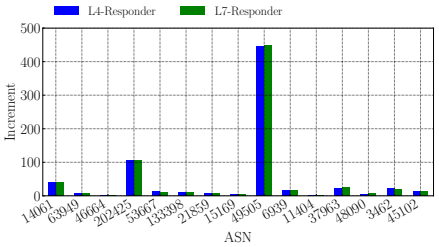


(f) Terminal

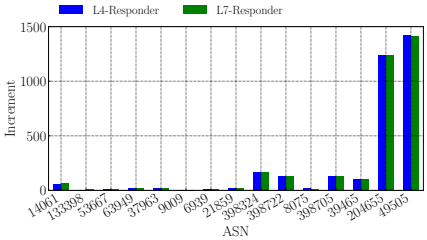


(g) Mail Server

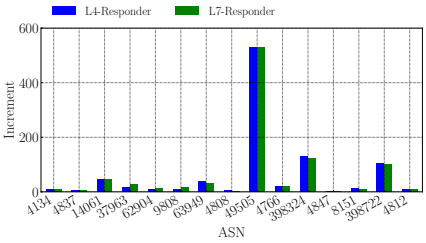
Figure 5.15: Increment in number of flows with respect to the Darknet of non-active services.



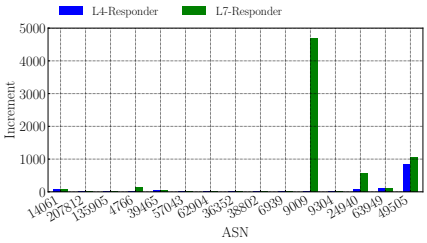
(a) Web Server



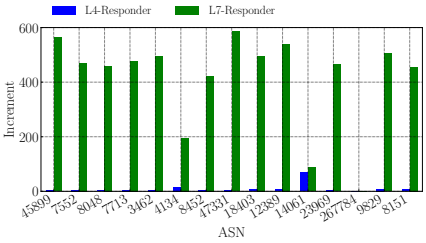
(b) Proxy



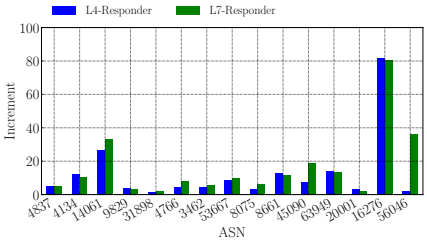
(c) DBserver



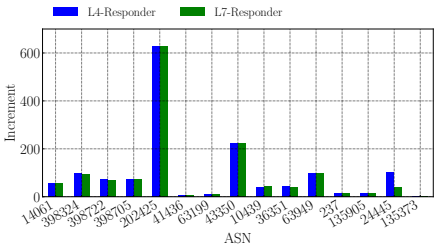
(d) Remote Desktop



(e) Fileserver



(f) Terminal



(g) Mail Server

Figure 5.16: Increment in number of flows per AS with respect to the Darknet per service.

5.5 Identifying protocols on-the-fly

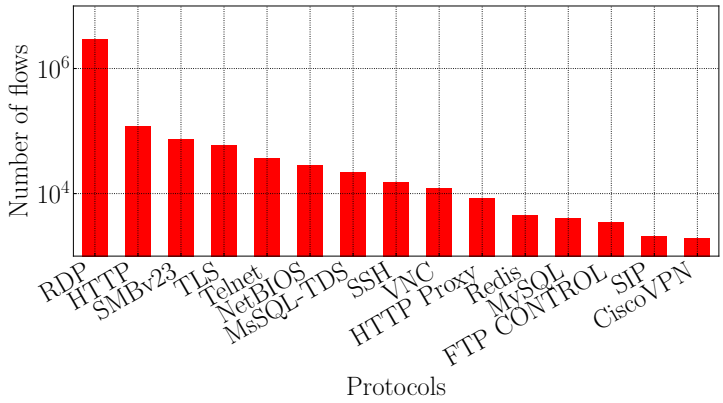
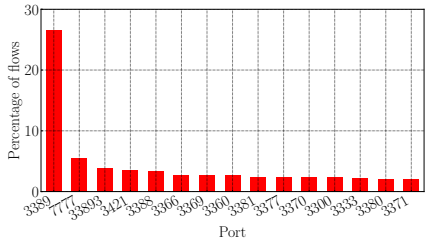


Figure 5.17: Top 15 protocols.

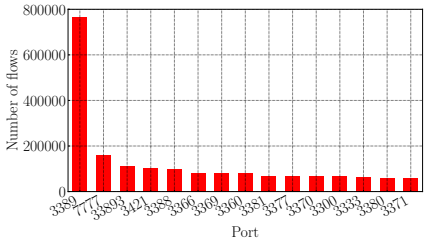
In this Section, we want to understand if by mean of the protocol identification on-the-fly on non-standard ports before replying, the attack patterns changes. Fig. 5.17 shows the number of flows for the top 15 protocols: the majority of flows are RDP, HTTP and SMB.

In Fig. 5.18 we report the absolute value and the percentage of flow per port for each service that are now supported by DPIpot, that are RDP, HTTP, TLS AND SSH. It is interesting to note that in 3 cases out of 4, the DPI protocol identification is useful to attract more traffic. Indeed, if we focus on RDP service, we see in Fig. 5.18(a) that only 25% of the RDP traffic is directed to the standard port 3389, while the remaining 65% is directed to non-standard ports. This consideration holds also for HTTP and TLS: in this case around 10% of the traffic is directed to the standard port, while the remaining 90% to non-standard ports. Finally, for what concerns SSH, more than 95% of the traffic is directed to the well-known port 22.

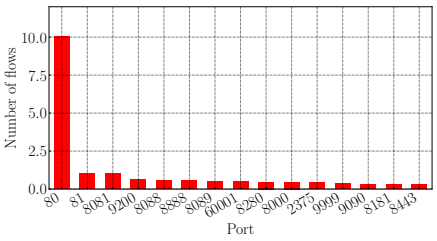
It turns out that if we start reply with the correct protocol we observe a significant increase in traffic not only on standard ports, but especially on non-standard ports, depending on the target protocol.



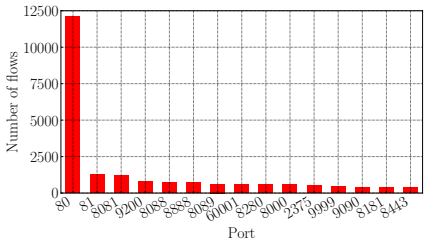
(a) RDP protocol percentage.



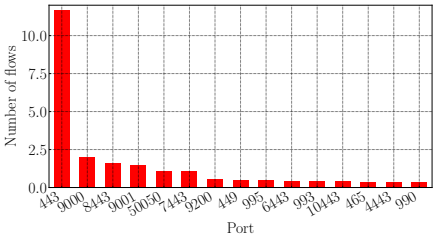
(b) RDP protocol



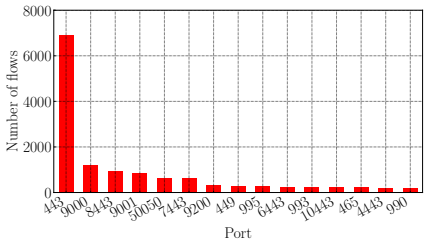
(c) HTTP protocol percentage.



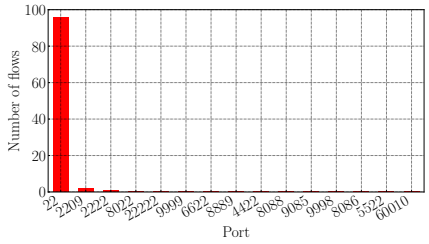
(d) HTTP protocol.



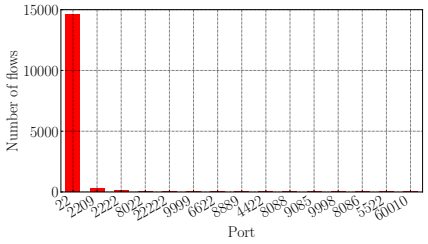
(e) TLS protocol percentage.



(f) TLS protocol.



(g) SSH protocol percentage.



(h) SSH protocol.

Figure 5.18: Top 15 ports for each DPIpot supported service.

Chapter 6

Conclusions and future works

6.1 Conclusions

In this thesis, firstly, we presented an evaluation of DPI solutions in several traffic scenarios, comparing the consistency of their classifications. The tools are practically equivalent when the input traffic is composed of popular and well-known protocols (e.g., HTTP, DNS and TLS). When applied to complex scenarios, such as to traffic generated by Malware scans, DPI tools struggle. In sum, there is space for improving these DPI tools by extending their label sets. Interestingly, tools reach steady-state classification after one packet, suggesting they can be exploited in online scenarios. Indeed, for our final purpose, we chose to exploit nDPI since it provides a good trade-off between accuracy and the number of supported protocols.

Then, we evaluated the impact of different deployments on the type of information obtained about attacks: we investigated whether these possible variations impact the type of information obtained by the honeypots. As a main result, we record an increment in traffic when we start replying. This increment is visible both in traffic volume and the number of remote sources reaching the infrastructure. We then observe that often, when the attacker finds an active service, it immediately starts probing adjacent ports.

When we expose only some services, we observe a significant increase only for those services. Furthermore, in most of the cases, there is no significant difference between replying at level four only or at layer seven only, except for some specific services, like RDP. Thus, in many cases, completing the three-way handshake is enough

to engage more attackers.

From the analysis of the states, we conclude that in all the considered scenarios there is a certain percentage of remote sources that only want to probe ports, and are not interested in establishing a connection. When we start replying, there is a clear increase of packets with payload: this type of traffic allows to gather more information about the attackers, and better understand how the attack is configured.

The analysis of the traffic reaching DPIpot leads to other interesting observations. We record that attacks on non-standard ports are very common for some specific services (i.e., RDP, HTTP or TLS), and replying with the correct protocol attracts a larger volume of attacks. For some other services, as in the SSH case, we see that the largest part of the traffic is instead directed to standard ports, so we can infer that using DPI does not contribute in a sensible way to the traffic patterns.

6.2 Future works

In the future, we plan to enlarge our observation window to a larger period of time, in order to validate the exiatence of patterns in time and to have more meaningful and stable results. We then want to extend the set of protocols supported by DPIpot to all the 100 protocols implemented by nDPI. Furthermore, we plan to improve the performance of DPIpot in order to support more connections simultaneously.

Bibliography

[1] Daniel Fraunholz, Simon Duque Anton, Christoph Lipps, Daniel Reti, Daniel Krohmer, Frederic Pohl, Matthias Tammen, and Hans Dieter Schotten. Demystifying deception technology: A survey. *arXiv preprint arXiv:1804.06196*, 2018.

[2] Julian Jang-Jaccard and Surya Nepal. A survey of emerging threats in cybersecurity. *Journal of Computer and System Sciences*, 80(5):973–993, 2014.

[3] Marcin Nawrocki, Matthias Wählisch, Thomas C Schmidt, Christian Keil, and Jochen Schönfelder. A survey on honeypot software and data analysis. *arXiv preprint arXiv:1608.06249*, 2016.

[4] Ajay Chaudhary and Anjali Sardana. Software based implementation methodologies for deep packet inspection. In *2011 international conference on information science and applications*, pages 1–10. IEEE, 2011.

[5] Michela Becchi, Mark Franklin, and Patrick Crowley. A workload for evaluating deep packet inspection architectures. In *2008 IEEE International Symposium on Workload Characterization*, pages 79–89. IEEE, 2008.

[6] Sailesh Kumar, Jonathan Turner, and John Williams. Advanced algorithms for fast and scalable deep packet inspection. In *2006 Symposium on Architecture For Networking And Communications Systems*, pages 81–92. IEEE, 2006.

[7] M. Trevisan, A. Finamore, M. Mellia, M. Munafo, and D. Rossi. Traffic Analysis with Off-the-Shelf Hardware: Challenges and Lessons Learned. *IEEE Commun. Mag.*, 55(3):163–169, 2017.

[8] Shane Alcock and Richard Nelson. Libprotoident: traffic classification using lightweight packet inspection. *WAND Network Research Group, Tech. Rep*, 2012.

[9] Luca Deri, Maurizio Martinelli, Tomasz Bujlow, and Alfredo Cardigliano. ndpi: Open-source high-speed deep packet inspection. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 617–622. IEEE, 2014.

[10] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[11] Marco Mellia, R Lo Cigno, and Fabio Neri. Measuring ip and tcp behavior on edge nodes with tstat. *Computer Networks*, 47(1):1–21, 2005.

[12] Andrew W Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In *International Workshop on Passive and Active Network Measurement*, pages 41–54. Springer, 2005.

[13] Jinghua Yan. A survey of traffic classification validation and ground truth collection. In *2018 8th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, pages 255–259. IEEE, 2018.

[14] S. Alcock and R. Nelson. Measuring the accuracy of open-source payload-based traffic classifiers using popular internet applications. In *38th Annual IEEE Conference on Local Computer Networks - Workshops*, pages 956–963, 2013.

[15] Géza Szabó, Dániel Orincsay, Szabolcs Malomsoky, and István Szabó. On the validation of traffic classification algorithms. In *International Conference on Passive and Active Network Measurement*, pages 72–81. Springer, 2008.

[16] Francesco Gringoli, Luca Salgarelli, Maurizio Dusi, Niccolo Cascarano, Fulvio Risso, and KC Claffy. Gt: picking up the truth from the ground for internet traffic. *ACM SIGCOMM Computer Communication Review*, 39(5):12–18, 2009.

[17] Peng Lizhi, Zhang Hongli, Yang Bo, Chen Yuehui, and Wu Tong. Traffic labeller: collecting internet traffic samples with accurate application information. *China Communications*, 11(1):69–78, 2014.

[18] Tomasz Bujlow, Valentín Carela-Español, and Pere Barlet-Ros. Independent comparison of popular dpi tools for traffic classification. *Computer Networks*, 76:75–89, 2015.

[19] Lance Spitzner. *Honeypots: tracking hackers*, volume 1. Addison-Wesley Reading, 2003.

[20] Daniel Fraunholz, Marc Zimmermann, and Hans D Schotten. An adaptive honeypot configuration, deployment and maintenance strategy. In *2017 19th International Conference on Advanced Communication Technology (ICACT)*, pages 53–57. IEEE, 2017.

[21] Warren Cabral, Craig Valli, Leslie Sikos, and Samuel Wakeling. Review and analysis of cowrie artefacts and their potential to be used deceptively. In *2019 International Conference on computational science and computational intelligence (CSCI)*, pages 166–171. IEEE, 2019.

[22] Armin Ziaie Tabari and Xinming Ou. A first step towards understanding real-world attacks on iot devices. *arXiv preprint arXiv:2003.01218*, 2020.

[23] Wenjun Fan, Zhihui Du, Max Smith-Creasey, and David Fernandez. Honeydoc: an efficient honeypot architecture enabling all-round design. *IEEE Journal on Selected Areas in Communications*, 37(3):683–697, 2019.

[24] L. Spitzner. The honeynet project: trapping the hackers. *IEEE Security Privacy*, 1(2):15–23, 2003.

[25] Tommaso Rescio, Thomas Favale, Francesca Soro, Marco Mellia, and Idilio Drago. DPI solutions in practice: benchmark and comparison. In *Proceeding of 6th International Workshop on Traffic Measurements for Cybersecurity*, Virtual Workshop, May 2021. 42nd IEEE Symposium on Security and Privacy.

[26] Hyunchul Kim, KC Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. Internet traffic classification demystified: Myths, caveats, and the best practices. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, New York, NY, USA, 2008. Association for Computing Machinery.

[27] G. Aceto, A. Dainotti, W. de Donato, and A. Pescape. Portload: Taking the best of two worlds in traffic classification. In *2010 INFOCOM IEEE Conference on Computer Communications Workshops*, pages 1–5, 2010.

[28] Jawad Khalife, Amjad Hajjar, and Jesús Díaz-Verdejo. Performance of opendpi in identifying sampled network traffic. *Journal of Networks*, 8(1):71, 2013.

[29] Antonio Nisticò, Dena Markudova, Martino Trevisan, Michela Meo, and Giovanna Carofiglio. A comparative study of rtc applications. *To appear in the Proceedings of the 22nd IEEE International Symposium on Multimedia*, 2020.

[30] Alberto Dainotti, Antonio Pescapé, and Giorgio Ventre. A packet-level characterization of network traffic. In *2006 11th International Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks*, pages 38–45. IEEE, 2006.

[31] Andrea Di Domenico, Gianluca Perna, Martino Trevisan, Luca Vassio, and Danilo Giordano. A network analysis on cloud gaming: Stadia, geforce now and psnow. *arXiv preprint arXiv:2012.06774*, 2020.

[32] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP*, pages 108–116, 2018.

[33] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. Classifying iot devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing*, 18(8):1745–1759, 2018.

[34] A Parmisano, S Garcia, and MJ Erquiaga. A labeled dataset with malicious and benign iot network traffic. *Stratosphere Laboratory: Praha, Czech Republic*, 2020.

[35] Marco Mellia, Idilio Drago, and Eros Filippi. Honeyport-a scalable meta-honeypot system for security applications, 2019.