Universität des Saarlandes

Towards Uncovering Hidden Internet Traffic Characteristics

Dissertation zur Erlangerung des Grades der
Doktorin der Ingenieurwissenschaften (Dr.-Ing.)
der Fakultät für Mathematik und Informatik der
Universität des Saarlandes

vorgelegt von

Aniss Maghsoudlou

Saarbrücken, 2023

Day of Colloquium	14 / 12 / 2023
Dean of the Faculty	Univ.-Prof. Dr. Jürgen Steimle
Chair of the Committee	Prof. Dr. Ingmar Weber
Reporters	Prof. Dr. Anja Feldmann
	Prof. dr. ir. Roland van Rijswijk-Deij
	Dr. Oliver Gasser
Academic Assistant	Dr. Ha Dao

Abstract

With the growing digitization of many dimensions of human life, including education, work, and entertainment, the Internet has become an inevitable utility. Since the Internet is designed and realized through the cooperation of many different organizations in a non-centralized manner with a best-effort mindset, it is essential to measure and monitor different aspects of the Internet including security, performance, and scalability. The rise of remote work has emphasized the need for measuring security of the Internet traffic.

In this thesis, we first address the need for monitoring and measuring large-scale Internet traffic to gain useful insights into the security and traffic trends in large Internet Service Providers (ISPs) and Internet eXchange Points (IXPs)–important building blocks of today's Internet–by designing a system called Flowyager for summarizing and querying network-wide flow data in a near real-time manner. This system enables network operators to spot Distributed Denial of Service (DDoS) attacks in the captured flow data and drill down specific parts of the flow data to gain more details about the attacks.

Next, we propose FlowDNS to augment flow data with domain names which gives us the chance to infer the actual service/domain to which the traffic belongs. This system lays the foundation for monitoring the services that are being used and gives network operators the chance to predict their bandwidth demands. The results from FlowDNS can later be combined with other sources of network data, e.g., routing data, to help network operators with their peering decisions.

FlowDNS and Flowyager provide means to analyze network traffic and find abnormal traffic behaviors on the existing packet captures. However, to gain a more comprehensive picture of Internet traffic, we need to combine the results from the above-mentioned systems with active measurement techniques. This gives us the chance to discover the existence and origin of hidden characteristics of the Internet traffic. For instance, in a large European ISP, we detect a large amount of Internet traffic using port number 0 when querying Flowyager. In this thesis, we complement passive measurement results with active measurement techniques to investigate the underlying causes of such traffic. We find that this traffic is mostly caused by fragmentation, scanning, and misconfigured devices.

Finally, given the widespread usage of Virtual Private Networks (VPNs) during the COVID-19 pandemic for remote work, we strive to characterize VPN traffic in the Internet. We use active measurement techniques to detect VPN servers and analyze their security aspects. Then, with the help of FlowDNS, we detect VPN traffic on the Internet to provide insights about the VPN traffic patterns in the Internet.

We publish the code for our systems, active measurements, and our analyses for future researchers to use. The works covered in this dissertation not only help researchers and network operators to gain insights about some hidden characteristics of Internet traffic but also provide the means to look for specific traffic patterns in the network flow data and investigate its characteristics.

Zusammenfassung

Mit der zunehmenden Digitalisierung vieler Bereiche des menschlichen Lebens, einschließlich Bildung, Arbeit und Unterhaltung, ist das Internet zu einem unverzichtbaren Hilfsmittel geworden. Da das Internet durch die Zusammenarbeit vieler verschiedener Organisationen in einer nicht zentralisierten Art und Weise mit einer Best-Effort-Mentalität entwickelt und realisiert wird, ist es unerlässlich, verschiedene Aspekte des Internets wie Sicherheit, Leistung und Skalierbarkeit zu messen und zu überwachen. Die Zunahme des Home Office hat die Notwendigkeit unterstrichen, die Sicherheit des Internet-Traffics zu messen.

In dieser Arbeit befassen wir uns zunächst mit der Notwendigkeit, den Internet-Traffic in großem Maßstab zu analysieren und zu messen, um nützliche Einblicke in die Sicherheits- und Traffic-Trends bei großen Internet Service Providern (ISPs) und Internet Exchange Points (IXPs) - wichtigen Bausteinen des heutigen Internets - zu gewinnen. Hierzu entwickeln wir ein System namens Flowyager zur Zusammenfassung und Abfrage von netzwerkweiten Datenströmen in nahezu Echtzeit. Dieses System ermöglicht es Netzbetreibern, DDoS-Angriffe (Distributed Denial of Service) in den erfassten Datenströmen zu erkennen und bestimmte Teile der Datenströme zu analysieren, um mehr Details über die Angriffe zu erfahren.

Als Nächstes schlagen wir FlowDNS vor, um Flow Data mit Domänennamen zu ergänzen. *Flow Data* bezeichnet hierbei die Informationen über einen Datenstrom, der sich über einen längeren Zeitraum zwei kommunizierenden Endstellen zuordnen lässt. Dadurch erhalten wir die Möglichkeit, auf den tatsächlichen Dienst/die Domäne zu schließen, zu dem/der die Datenströme gehören. Dieses System gibt Netzbetreibern die Möglichkeit, ihren Bandbreitenbedarf vorherzusagen. Die Ergebnisse von FlowDNS können später mit anderen Quellen von Netzdaten, z.B. Routing-Daten, kombiniert werden, um Netzbetreibern bei ihren Peering-Entscheidungen zu helfen.

FlowDNS und Flowyager bieten die Möglichkeit, die Netzwerkdatenströme zu analysieren und abnormale Traffic-Muster anhand der vorhandenen Daten zu erkennen. Um jedoch ein umfassenderes Bild des Internet-Traffics zu erhalten, müssen wir die Ergebnisse der oben genannten Systeme mit aktiven Messverfahren kombinieren. Dies gibt uns die Möglichkeit, die Existenz und den Ursprung verborgener Merkmale des Internet-Traffics zu entdecken. So stellen wir beispielsweise bei einem großen europäischen ISP fest, dass ein großer Teil des Internet-Traffics über die Portnummer 0 abgewickelt wird. Deshalb ergänzen wir die passiven Messergebnisse mit aktiven Messverfahren, um die Ursachen für diese Datenströme zu untersuchen. Wir stellen fest, dass diese Datenströme hauptsächlich durch Fragmentierung, Scannen und falsch konfigurierte Geräte verursacht werden.

Angesichts der weit verbreiteten Nutzung Virtueller Privater Netzwerke (VPNs) während der Corona-Pandemie für das Home Office untersuchen wir die VPN-Datenströme im Internet zu charakterisieren. Hierzu verwenden wir aktive Messverfahren, um VPN-Server aufzuspüren und ihre Sicherheitsaspekte zu analysieren. Anschließend

ermitteln wir mit hilfe von FlowDNS die VPN-Datenströme im Internet, um Erkenntnisse über deren Muster im Internet zu gewinnen.

Wir veröffentlichen den Code für unsere Systeme, die aktiven Messungen und unsere Analysen, damit zukünftige Forscher sie nutzen können. Die in dieser Dissertation behandelten Arbeiten helfen nicht nur Forschern und Netzbetreibern, Einblicke in einige verborgene Merkmale des Internet-Traffics zu gewinnen, sondern bieten auch die Möglichkeit, nach bestimmten Mustern der Internet-Datenströme zu suchen und deren Merkmale zu untersuchen.

Acknowledgements

Pursuing a PhD is known to be challenging in different aspects, starting from research complexity and sometimes setbacks to isolation, disappointment, and work-life balance. However, I feel incredibly fortunate to have had the privilege to be surrounded by so many supporting individuals that mitigated these challenges.

First and foremost, I would like to extend my sincere appreciation to my advisor, Anja Feldmann. Her understanding, guidance, and invaluable mentorship have shaped my academic journey. I am grateful for the lessons I have learned from her, not only in terms of professional growth but also in caring for others while maintaining productivity.

I am immensely grateful to my husband, Mohamad, for his unwavering support throughout my PhD journey. His constant encouragement, understanding, and assistance during deadlines and the highs and lows of academic life have been instrumental. I am also thankful for his unwavering support over the past decade and during our immigration journey to Germany.

My heartfelt thanks also go to Oliver, my major collaborator and supervisor in most of my work. His emphasis on discipline and attention to details has greatly influenced my approach to research. I would also like to acknowledge Jawad, my first collaborator at INET, for his valuable assistance and guidance, showing me the ropes and helping me understand how things work.

I am grateful to Seif, Emilia, Lars, Danesh, and Fariba for their friendship and the meaningful exchanges of thoughts throughout my PhD journey. I would like to thank Florian, Franziska, Mirko, Ali, Daniel, Victor, Cristi, Pascal, and the entire INET team, for their constructive feedbacks, and for making INET an excellent place for research.

Last but not least, I am deeply indebted to my parents for their constant motivation and support. Their belief in my educational pursuits and their presence, whenever I needed them, have been fundamental to my achievements.

I extend my sincere gratitude to all those mentioned above and to anyone else who has contributed to my PhD journey. Your support, guidance, and encouragement have been invaluable, and I am truly grateful.

Publications

Published Papers

Parts of this thesis are based on the following peer-reviewed papers that have already been published. All my collaborators are listed among the co-authors.

International Conferences

Aniss Maghsoudlou, Oliver Gasser, and Anja Feldmann. "Zeroing in on Port 0 Traffic in the Wild". In: *Passive and Active Measurement*. Ed. by Oliver Hohlfeld, Andra Lutu, and Dave Levin. Cham: Springer International Publishing, 2021, pp. 547–563. ISBN: 978-3-030-72582-2

Aniss Maghsoudlou, Oliver Gasser, Ingmar Poese, and Anja Feldmann. "FlowDNS: Correlating Netflow and DNS Streams at Scale". In: *Proceedings of the 18th International Conference on Emerging Networking EXperiments and Technologies*. CoNEXT '22. Roma, Italy: Association for Computing Machinery, 2022, pp. 187–195. ISBN: 9781450395083

Aniss Maghsoudlou, Lukas Vermeulen, Ingmar Poese, and Oliver Gasser. "Characterizing the VPN Ecosystem in the Wild". In: *Passive and Active Measurement*. Ed. by Anna Brunstrom, Marcel Flores, and Marco Fiore. Cham: Springer Nature Switzerland, 2023, pp. 18–45. ISBN: 978-3-031-28486-1

Peer-reviewed Journals

Said Jawad Saidi, Aniss Maghsoudlou, Damien Foucard, Georgios Smaragdakis, Ingmar Poese, and Anja Feldmann. "Exploring Network-Wide Flow Data With Flowyager". In: *IEEE Transactions on Network and Service Management* 17.4 (2020), pp. 1988–2006

Workshops and Poster Sessions

Aniss Maghsoudlou, Oliver Gasser, and Anja Feldmann. *Reserved: Dissecting Internet Traffic on Port 0*. Poster at Passive and Active Measurement Conference 2020. 2020

Contents

1

Introduction

The Internet was initially built to share information between a limited number of known hosts used for a few applications. It then gradually evolved into its current form connecting 5 billion users around the globe as of January 2023 [6]. Internet measurement has been in place since the birth of the Internet for debugging purposes, deployment planning, and future developments, and it still is one of the critical requirements of the Internet. The evolution of the Internet was not led or planned by a single organization with built-in measurement mechanisms, therefore, measuring such a distributed system is needed continuously not only to understand how it is evolving but also to shape its future. Also, Internet failures can still be caused intentionally or by mistake, keeping a specific service or user group from the Internet, e.g., the Facebook outage in 2021 [7], or the Verizon outage in 2019 [8]. Internet measurements help reduce these failures and faciliate building counter-measures to recover from them as soon as possible. Moreover, service providers need to ensure that their infrastructure and network resources match with the traffic demands requiring Internet measurements to take place.

Internet measurements assist Internet Service Providers in gaining insights into their networks, and enabling them to engineer their networks to meet customers' demands and make informed peering desicions. For traffic engineering, ISPs need to know their customers' traffic characteristics including the traffic volume, the origins or destinations of their traffic, and the applications or services their customers are using. In addition, Internet traffic characteristics are of great importance due to security concerns. In case of a cyber attack to a service provider infrastucture or a customer network, knowing the characteristics of the attack is crucial not only to measure the scope of the attack, but also to avoid it in the future.

Another purpose of Internet measurement is to assess compliance with Internet protocols and standards. Although the Internet has evolved organically and protocols are designed and deployed on-demand, there are standards enforcing certain rules on those protocols to enable hosts around the world to communicate. These limitations, however, are not always respected, either intentionally, or due to misconfigurations. Not following the standards or commonly expected behaviors can bring network devices or mechanisms into unexpected states and therefore might induce threats to networks. Therefore, it is important to measure Internet traffic from the standards compliance perspective as well.

However, extracting important characteristics from Internet traffic is challenging, first, due to the sheer amount of traffic in large networks, and second, due to the widespread use of encrypted traffic.

Several terabytes of traffic pass service provider networks every day. These service providers capture and store certain information about the traffic, e.g., the source and destination address and port number identifying a traffic flow. Storing such a huge amount of traffic flows as-is is impossible and the analysis needs to be done on an excerpt of the traffic. Also, to be able to adapt to changes as fast as possible, the analysis needs to be done as quickly as possible. There are a substantial number of studies that address the problem of analyzing network traffic captures and answering queries about the health and load of the system. However, surprisingly, there are very few studies that enable issuing queries that are a priori unknown. In most of the studies, network operators need to issue the query and then wait for that query to be applied to the incoming data, and no new query can be applied to the historic data which limits the functionality of such a system greatly. Therefore, the first research question I plan to cover in this dissertation is the following: **How can we monitor the traffic using the existing network flow captures in near real-time with a priori unknown queries?**

Useful information such as the source and destination of the traffic is exposed to the ISPs, therefore, most ISPs nowadays collect flow information on their interfaces. This information not only helps them with better planning of their networks but also yields fault detection benefits in certain scenarios. However, this information is sometimes not enough on its own to grasp a precise picture of the traffic. Other metrics such as the corresponding domain name are needed for knowing customers interests and traffic classification. However, the domain name information is usually encrypted and therefore, not exposed to the ISP. Therefore, the second research question I am going to answer is the following: **How can we recognize the application (domain name) a certain traffic flow is using?**

Traffic monitoring helps network operators find the volume and direction of the traffic in their networks. However, it might not reveal certain aspects of the traffic and the reason behind some abnormalities. For instance, researchers have shown that there is a non-negligible share of traffic using port number 0 in the Internet [9, 10], although this port number cannot be used according to the IANA protocol port number registry [11]. Therefore, the third research question I strive to answer in this dissertation is the following: **Why do we see port 0 in the Internet and how can we characterize this traffic?**

With the COVID-19 outbreak and the increasing need to work remotely, people are using Virtual Private Networks (VPNs) to securely connect to their companies networks. Therefore, their traffic is encrypted and the network operators cannot distinguish this kind of traffic. Therefore, it is important to investigate the following research question: **How can we characterize the encrypted VPN traffic?**

Throughout this dissertation, I address these challenges and try to uncover hidden characteristics of Internet traffic. To accomplish this, I (a) present a system designed

for large-scale Internet traffic monitoring to extract, store, and query the most important traffic characteristics including the originating and destination addresses of the heavy hitters, (b) present a system designed for large scale correlation of different data sources, namely DNS and Netflow, to uncover the services behind Internet traffic flows, (c) use the aforementioned systems along with active measurements to uncover an illegitimate category of traffic and its intents, and (d) use the passive measurement along with active measurements to characterize encrypted traffic flows and detect VPN traffic.

Note that throughout this dissertation, I use the pronoun *we* instead of *I* wherever the work is done in collaboration with multiple authors.

1.1 Contributions

As the overarching goal of this thesis was to characterize Internet traffic, we made use of a broad range of Internet traffic datasets, including publicly available datasets, and Internet traffic from large European ISPs and IXPs. To process such large datasets, we built systems to be able to query network-related characteristics, and in some cases, we made use of multiple methodologies to know more about specific characteristics of Internet traffic.

Traffic monitoring. Many network operations, ranging from attack investigation and mitigation to traffic management, require answering network-wide flow queries in seconds. Although flow records are collected at each router, using available traffic capture utilities, querying the resulting datasets from hundreds of routers across sites and over time, remains a significant challenge due to the sheer traffic volume and distributed nature of flow records. In this contribution, we investigate how to improve the response time for *a priori unknown network-wide* queries. In Chapter 3, we present Flowyager, a system that is built on top of existing traffic capture utilities. Flowyager generates and analyzes tree data structures, which we call Flowtrees, which are succinct summaries of the raw flow data available by capture utilities. Flowtrees are self-adjusted data structures that drastically reduce space and transfer requirements, by 75% to 95%, compared to raw flow records. Flowyager manages the storage and transfers of Flowtrees, supports Flowtree operators, and provides a structured query language for answering flow queries across sites and time periods. By deploying a Flowyager prototype at both a large Internet Exchange Point and a Tier-1 Internet Service Provider, we showcase its capabilities for networks with hundreds of router interfaces. The results show that the query response time can be reduced by an order of magnitude when compared with alternative data analytics platforms. Thus, Flowyager enables *interactive network-wide queries* and offers unprecedented drill-down capabilities to, e.g., identify DDoS culprits, pinpoint the involved sites, and determine the length of the attack.

Domain name recognition. Knowing customer's interests, e.g., which Video-On-Demand (VoD) or Social Network services they are using, helps telecommunication companies with better network planning to enhance the performance exactly where

the customer's interests lie, and also offer the customers relevant commercial packages. However, with the increasing deployment of CDNs by different services, identification, and attribution of the traffic on network-layer information alone becomes a challenge: If multiple services are using the same CDN provider, they cannot be easily distinguished based on IP prefixes alone. Therefore, it is crucial to go beyond pure network-layer information for traffic attribution.

In Chapter 4, we leverage real-time DNS responses gathered by the clients' default DNS resolvers. Having these DNS responses and correlating them with network-layer headers, we are able to translate CDN-hosted domains to the actual services they belong to. We design a correlation system for this purpose and deploy it at a large European ISP. With our system, we can correlate an average of 81.7% of the traffic with the corresponding services, without any loss on our live data streams. Our correlation results also show that 0.5% of the daily traffic contains malformatted, spamming, or phishing domain names. Moreover, ISPs can correlate the results with their BGP information to find more details about the origin and destination of the traffic. We plan to publish our correlation software for other researchers or network operators to use.

Standards compliance: Port 0. Internet services leverage transport protocol port numbers to specify the source and destination application layer protocols. While using port 0 is not allowed in most transport protocols, we see a non-negligible share of traffic using port 0 in the Internet. In Chapter 5, we dissect port 0 traffic to infer its possible origins and causes using five complementing flow-level and packet-level datasets. We observe 73 GB of port 0 traffic in one week of IXP traffic, most of which we identify as an artifact of packet fragmentation. In our packet-level datasets, most traffic is originated from a small number of hosts and while most of the packets have no payload, a major fraction of packets containing payload belong to the BitTorrent protocol. Moreover, we find unique traffic patterns commonly seen in scanning. In addition to analyzing passive traces, we also conduct an active measurement campaign to study how different networks react to port 0 traffic. We find an unexpectedly high response rate for TCP port 0 probes in IPv4, with very low response rates with other protocol types. Finally, we will be running continuous port 0 measurements and providing the results to the measurement community.

Characterizing the encrypted traffic. With the increase of remote working during and after the COVID-19 pandemic, the use of Virtual Private Networks (VPNs) around the world has nearly doubled. Therefore, measuring the traffic and security aspects of the VPN ecosystem is more important now than ever. VPN users rely on the security of VPN solutions, to protect private and corporate communication. Thus a good understanding of the security state of VPN servers is crucial. Moreover, properly detecting and characterizing VPN traffic remains challenging, since some VPN protocols use the same port number as web traffic and port-based traffic classification will not help.

In Chapter 6, we aim at detecting and characterizing VPN servers in the wild, which facilitates detecting the VPN traffic. To this end, we perform Internet-wide active measurements to find VPN servers in the wild, and analyze their cryptographic

certificates, vulnerabilities, locations, and fingerprints. We find 9.8M VPN servers distributed around the world using OpenVPN, SSTP, PPTP, and IPsec, and analyze their vulnerability. We find SSTP to be the most vulnerable protocol with more than 90% of detected servers being vulnerable to TLS downgrade attacks. Out of all the servers that respond to our VPN probes, 2% also respond to HTTP probes and therefore are classified as Web servers. Finally, we use our list of VPN servers to identify VPN traffic in a large European ISP and observe that 2.6% of all traffic is related to these VPN servers.

1.2 Published Papers

This dissertation is based on multiple published papers in collaboration with many authors. In this section, I outline the main contributions of the thesis author.

Flowyager: Exploring Network-Wide Flow Capture Data

This work, explained in Chapter 3, is published in IEEE TNSM in collaboration with other co-authors [4]. The author's main contributions are (a) a survey of typical network queries and systems to tackle them. (b) joint design of FlowQL query language. (c) implementation and evaluation of the FlowQL module. (d) joint data visualization, plotting, and writing.

Zeroing in on Port 0 Traffic in the Wild

This work, explained in Chapter 5, is published in PAM 2021 proceedings in collaboration with other co-authors [1] and is also presented as a poster [5] at PAM Conference 2020. The author's main contributions are (a) formulation of the research question, (b) design and development of the active measurement setup, (c) investigating port 0 behavior both by analyzing passive datasets and active measurement results.

FlowDNS: Correlating Netflow and DNS Streams at Scale

This work, explained in Chapter 4, is submitted in collaboration with other co-authors in [2]. The author's main contributions are (a) the design, prototyping, and development of the FlowDNS system, (b) the deployment of FlowDNS in two European ISPs, and (c) evaluation of the proposed system.

Characterizing the VPN Ecosystem in the Wild

This work, explained in Chapter 6, is submitted in collaboration with other co-authors [3]. The author's main contributions are (a) formulation of the research question, (b) performing active measurements for VPN ports that do not require a handshake, i.e., OpenVPN, and IPsec, (c) detecting and investigating the VPN traffic in a large European ISP, and (d) comparing the detected VPN traffic volume to the state-of-the-art methods.

1.3 Thesis Structure

I structure this dissertation as follows: In Chapter 2, I go through the relevant information on transport layer ports, DNS concept, basics of VPN, and measurement methods to prepare a foundation on which the next chapters are built. Chapter 3 highlights the challenges in analyzing the existing flow data with a priori unknown queries and proposes a lightweight tree structure, called Flowtree to keep the summarized network flow data. Then we introduce a system, called Flowyager using this data structure together with a query language, namely, FlowQL to address the existing challenges. We finally compare the proposed system with the existing solutions. In Chapter 4, we propose a system, namely FlowDNS to correlate DNS and Netflow data in real-time which facilitates domain name recognition. We measure the resource usage of the system, then we go over some use cases for such a system. In Chapter 5, we analyze Internet traffic using port number 0 using the aforementioned systems in addition to active measurements to specify the origins and intents of this traffic. In Chapter 6, using FlowDNS, Flowyager, and active measurements, we analyze Internet traffic from an ISP, and try to distinguish between encrypted web traffic and VPN traffic. Finally, Chapter 7 summarizes the main points and findings of this dissertation, and discusses the future work.

2

Background

The Internet has altered the world in many ways, making innumerable applications possible, from working remotely to communicating with our loved ones from thousands of miles away. The plethora of applications brought about by the Internet has made it economically and politically important. Therefore, monitoring and measuring such an important network is vital.

This chapter provides an overview of the essential background information necessary to lay the foundation for the works in the following chapters. First, to give an overview of how the Internet works, I will introduce the Internet protocols, including the Domain Name System (DNS), which will be the primary focus of Chapter 4. Next, I will discuss the Internet structure and how different Internet organizations, such as ISPs, IXPs, and CDNs interact to provide end-customers with the Internet. Furthermore, I will emphasize the importance of closely investigating traffic flowing through these organizations. Finally, I will describe the necessity of Internet measurement and discuss different approaches for conducting such measurements.

2.1 Internet Protocols

The Internet today uses the Internet protocol suite which includes a set of communication protocols organized in layers. Link layer protocols define the networking methods in the scope of local network links and transmission of the frames to next-neighbor hosts. As soon as these frames leave the local network, network layer protocols such as IP help the packets navigate through the Internet. In the case of Internet Protocol (IP), IP addresses are added to these packets to help the routers find suitable routes for the packets. Transport layer protocols provide the communication channel between the applications in the end hosts. In other words, they help determine the exact application which is sending/receiving the packet in the end hosts, along with flow control and connection establishment. The most common transport layer protocols in the TCP/IP protocol stack. i.e., TCP and UDP, use a set of standardized port numbers to specify the application the packets belong to. The application layer protocols use the above-mentioned layers to communicate user data with the other hosts. Among the application layer protocols are HTTP, SSH, and DNS. DNS associates numerical IP addresses with more human-readable domain names. It uses a client-server model, with servers being the domain name resolvers. Each domain

name has at least one authoritative DNS server that is responsible for resolving the assignments for that domain name and any sub-domain of that domain.

Consider a scenario in which a client wants to access a web page through HTTP. The client's browser knows that HTTP uses port number 80, and it is given the domain name that needs to be accessed, assume *example.de* is the domain name.

However, what remains unknown to the client's browser is the IP address corresponding to the web server serving the desired web page. To acquire the IP address of the web server given the domain name, DNS is needed. Figure 2.1 shows how DNS works. In this case, the client's browser needs to send a DNS query, containing the domain name *example.de* to the local DNS resolver residing on the client's machine or local network. Then the local resolver takes on the rest of the process. It first sends the DNS query for domain name *example.de* to a DNS resolver according to its configurations.

The DNS resolver configured in the browser receives the DNS query for *example.de*. If the resolver finds an IP associated with *example.de* in its cache, it returns the IP to the browser. If no IP is found in the cache, the DNS resolver looks for the authoritative server for *.de*. If no match is found, it queries the root DNS server for *.de* to find the authoritative server for *.de*. Then, the DNS resolver queries the authoritative server for *.de* to find the authoritative DNS server for *example.de*. Next, the authoritative server for *example.de* is queried and the IP associated with *example.de* is returned to the client (i.e., the browser). Now the browser can use the returned IP address to establish the HTTP connection.

It uses the returned IP address as the destination IP address to send the first HTTP GET request. Then the Web server for *example.de* sends the landing page back to the browser.

I used HTTP as an example only for the sake of simplicity, however in today's Internet, with the increasing number of attacks and eavesdropping on Internet traffic, encrypting and securing this traffic becomes more and more important. Nowadays, users not only use HTTPS (HTTP Secure) for their Web traffic to the Web servers offering HTTPS but also use Virtual Private Networks (VPNs) to secure all their traffic. VPNs help secure the traffic by encrypting and tunneling them. VPN is also used to remotely access specific resources located in a company's infrastructure or to circumvent censorship. There are many different VPN protocols including SSTP, PPTP, IPsec, OpenVPN, and Wireguard. All these protocols establish a secure connection using a set of cipher suites between the client and server, then send/receive the data in encrypted form using the established secure channel. Since VPNs are commonly used for remote work, many researchers have focused on the VPN traffic specifically after the COVID-19 pandemic [12, 13].

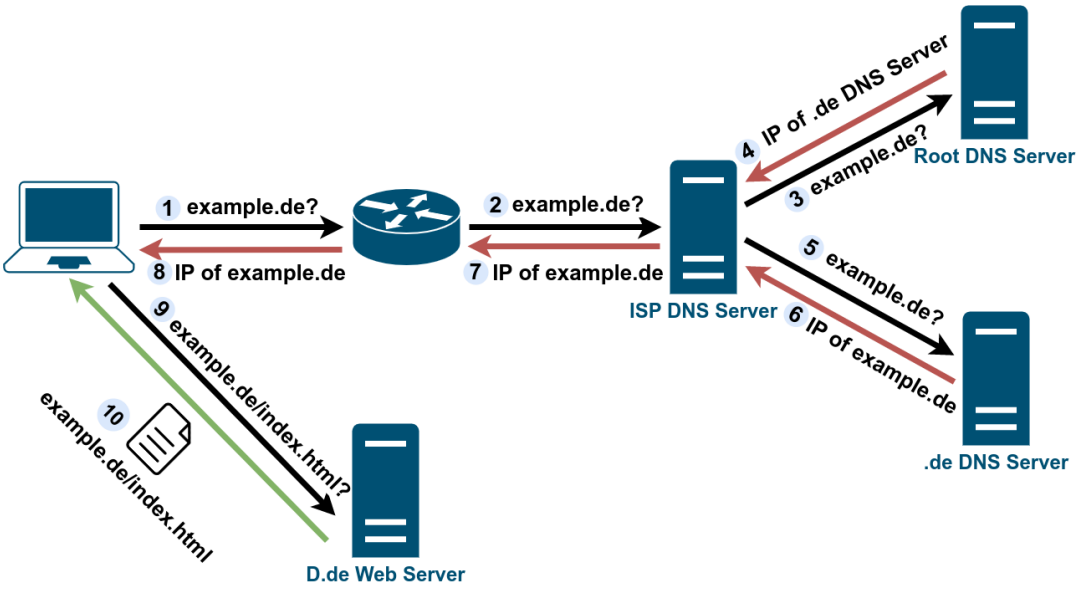


Figure 2.1: How Domain Name System Works

2.2 Internet Structure

The current Internet is made possible with the interconnection of Internet Service Providers (ISPs), Internet eXchange Points (IXPs), backbone infrastructure, and large content providers and Content Delivery Networks (CDNs). Figure 2.2 shows the topology of the dominant Internet traffic patterns [14]. Most of the modern-day Internet traffic is exchanged between large content providers, CDNs, and consumer networks. In this section, we introduce these important Internet organizations.

2.2.1 Internet Service Providers

Internet Service Providers are organizations that provide services for accessing the Internet. ISPs can be categorized into different categories based on their services and infrastructure. The ISP 3-tier model categorizes ISPs into three main categories [15]:

- **Tier 1 ISPs**. Networks that function as the backbone of the Internet, and therefore, also called backbone Internet providers. These ISPs usually build, own, and manage their own infrastructure such as the Trans-Atlantic undersea cables. Tier 1 ISPs include Deutsche Telekom, British Telekom, AT&T, etc. They exchange traffic with other Tier 1 ISPs via free peering interconnections, and usually get paid by Tier 2 ISPs, Tier 3 ISPs, and their own subscribers.
- **Tier 2 ISPs**. Regional or national providers that pay Tier 1 ISPs to exchange traffic with them, and also peer with other Tier 2 ISPs to increase their con-

nectivity. They get paid by the Tier 3 ISPs to provide them with connectivity to the Internet, and also by their own subscribers.
- **Tier 3 ISPs**. Providers that strictly purchase Internet transit from higher-tier ISPs for accessing the Internet, and are mainly involved in providing Internet access to end consumers. Therefore, Tier 3 ISPs are also referred to as consumer networks. These ISPs usually also offer local DNS resolvers that can be configured as the default DNS resolvers by end users.

As explained above, different Tiers of ISPs have different business models. Some are in direct contact with the end-users and some provide service for other Internet organizations. In either case, they need to be able to keep a close eye on the amount of traffic they exchange with other organizations or for end-users. Therefore, it is necessary for them to monitor their network traffic. We extensively address this in Chapter 3. Also, we analyze different characteristics of the traffic from ISPs throughout this thesis, including their DNS data in Chapter 4, the existence of traffic using port 0 in Chapter 5, and detecting VPN traffic among ISP data in Chapter 6.

2.2.2 Internet Exchange Points

The interconnection, or peering, of ISPs at different levels is the fundamental building block of the Internet's structure. Therefore, network exchange facilities came into existence providing a physical infrastructure and neutral gathering point where ISPs or other networks can exchange traffic with each other, independent of third parties [16]. These facilities are called Internet eXchange Points (IXPs) in the modern Internet terminology. Connecting to an IXP helps companies to shorten their path to the transit and therefore, reduces latency, improves round-trip time, and potentially reduces costs [17]. In a large European IXP, petabytes of traffic are exchanged every day, therefore, there is a large amount of data that needs to be processed for security and monitoring purposes. This challenge has been addressed in Chapter 3. Moreover, IXPs are very important as a vantage point since the traffic comes from many different sources and is destined to many different destinations. Figure 2.2 shows how IXPs help the interconnection of the ISPs in different levels.

Therefore, investigating the traffic going through IXPs can be helpful to uncover hidden characteristics of Internet traffic, an instance of which we study in Chapter 5 with the use of IXP data.

2.2.3 Internet Content Providers

Any website or organization that manages access to a content repository through the Internet is referred to as Internet Content Provider (ICP) or generally, content provider. Content providers can be categorized by the type of content, including but not limited to media, news, forums, and blogs, or by their business models. Some content providers charge the end-users directly on a regular or on-demand basis, while others might show them advertisements, or a combination of the two. In either case,

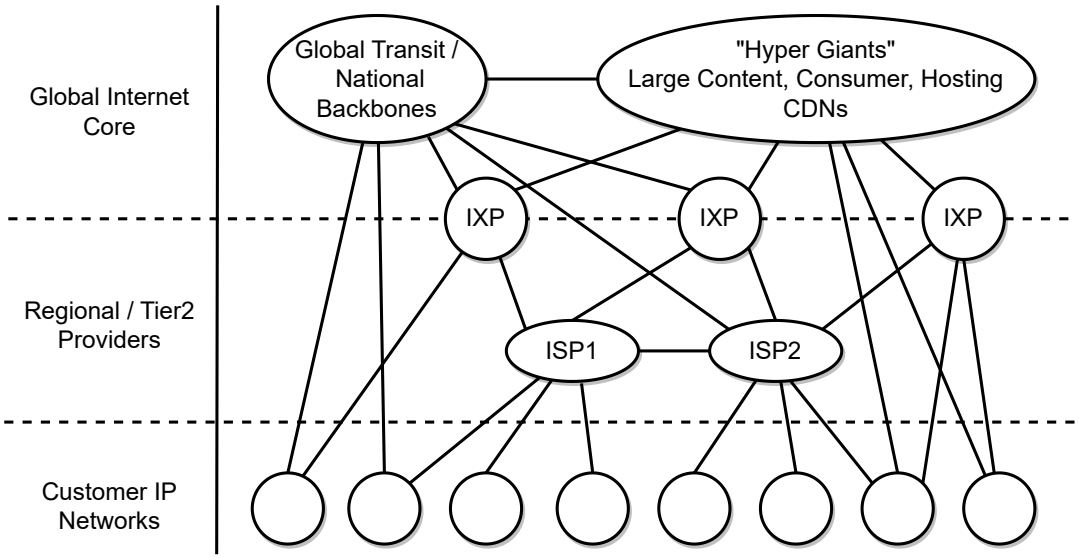


Figure 2.2: Overview of the Internet structure borrowed from Labovitz et al. [14]

one important task for the ICPs is to deliver their content to the end users via the Internet, and this can be challenging due to different reasons. They need to make sure that the content is delivered with the least delay, is reliably accessible, and is safe against cyber attacks. Therefore, ICPs sometimes use a CDN to take care of a timely, reliable, and secure content delivery.

2.2.4 Content Delivery Networks

A Content Delivery Network (CDN) consists of a network of servers spread across multiple locations. These servers collaborate to deliver Internet content fast, reliably, and safely. Content providers such as video streaming services and news media use CDNs not only to bring their content closer to their customers and increase their website load times but also to increase content availability and redundancy and website security. Some content providers use third-party CDNs to deliver their content such as Twitch, a streaming platform widely used by gamers, and Vimeo, a video-sharing website, which both currently use a combination of multiple third-party CDNs. In the meanwhile, some companies implement and use their own CDNs such as Google, Amazon, and Netflix to deliver their content. As Figure 2.2 suggests, CDNs can directly connect to IXPs and different tiers of ISPs according to their policies and size. Most of the large content providers use CDNs for delivering their content, therefore, traffic coming from CDNs contributes to a large fraction of Internet traffic and is therefore interesting for network operators to monitor. In Chapter 4, we provide a system design to be able to categorize the traffic specifically coming from CDNs with more details. Big CDNs and content providers tend to directly connect to ISPs that are in different tiers to enhance their performance.

2.3 Internet Measurement

Almost all the protocols commonly used in the Internet are standardized by the Internet Engineering Task Force (IETF) as Request For Comments (RFC). However, these documents only specify details and rules on how to implement and use a protocol. Therefore, there can be different implementations of the same protocol and Internet measurements are needed to measure their compliance with the standards and to investigate how the deviation from the standard can effect the Internet. On the other hand, Internet protocols evolve over time and it is challenging to understand how the evolution of the Internet protocols and the interplay between differrent Internet protocols affect users and networks. Internet measurements, i.e., careful investigation of Internet traffic and Internet protocols from different aspects, enables the research community and network operators to gain insights into these issues. Network operators also need to know about the traffic being exchanged in their networks for evaluating the security of their networks, use of the resources, traffic shaping, and invoicing purposes, which makes network monitoring an essential task.

To be able to extract meaningful information from the above-mentioned perspectives, one needs to look at the right sources of traffic. Therefore, in our contributions to uncover traffic characteristics, we use two different sources of data:

(1) active measurement data: we analyze the traffic we generate targeting the whole Internet address space and extract characteristics of this traffic.

(2) passive measurement data: we analyze the traffic going through major IXPs and ISPs in Europe to analyze the traffic in its most realistic form. Throughout the thesis, we frequently use traffic data from a large European IXP and a large European ISP, and occasionally publicly available traffic datasets, namely MAWI and CAIDA.

Using multiple sources of traffic along with active measurements gives us a relatively unbiased perspective into the actual characteristics of Internet traffic. In rest of this section, we explain how we perform active and passive measurement with more details.

2.3.1 Passive Measurement

Passive measurement is the process of keeping track of network traffic adding any traffic or affecting the existing one [18]. The advantage of such an approach is that it measures real traffic although the measurement metrics need to be determined in advance. In addition, using passive measurement usually requires processing large volumes of data and therefore, requires specific methods to aggregate network data. Passive measurement is a common approach used by network operators to gain information about various characteristics of the network traffic. ISPs and IXPs usually capture their traffic information, usually sampled, in a pre-determined format, such as Netflow or IPFIX and analyze them for traffic profiling, intrusion detection, and so on. The 5-tuple, namely a combination of source IP address, destination IP address,

source port, destination port, and protocol are among the frequently gathered flow information in such flow recordings.

Netflow. NetFlow is a feature originally introduced in Cisco routers to collect IP network traffic on an interface. The packets entering or exiting one interface are aggregated into *flows* by the flow exporter component of the router, then the flows are received by the flow exporters to be stored and pre-processed, waiting for further analysis. Netflow version 5 defines a *flow* as a uni-directional sequence of packets sharing the following values:

- Ingress interface: the interface ID on which the packet is received.
- Source IP address
- Destination IP address
- IP protocol: IPv4 or IPv6
- Source port for UDP and TCP, or 0 for other protocols
- Destionation port for UDP and TCP, type and code for ICMP, or 0 for other protocols
- IP Type of Service (ToS)

While Netflow version 5 only allows for exporting specific fields, the most recent NetFlow version 9 allows for template-based flow definitions introducing more flexibility. Since capturing all the packets are too costly for big networks such as those of ISPs and IXPs, flow exporters usually sample one out of n packets depending on the configurations.

IPFIX. Inspired by NetFlow, the IETF community came up with an open standard format for capturing networks flow information, called IPFIX, which is now widely used by many different networking vendors apart from Cisco. While IPFIX and NetFlow are very similar in nature, IPFIX, unlike NetFlow, also allows for variable length fields.

Throughout this thesis, the author extensively uses passive measurements to characterize Internet traffic and propose systems to facilitate processing the large volume of passive captures.

2.3.1.1 Ethical Considerations

When conducting passive measurement for the research in this dissertation, we comply with the NDAs and ensure that no personally identifiable information is shared when analyzing passive flow and packet data. Unlike active measurements, we will not publish any of the passive measurement data. In the research presented in Chapters Chapter 3, Chapter 4, Chapter 5, and Chapter 6, we process all ISP and IXP data on their own premises and do not copy, transfer, or store any data outside of the servers dedicated to NetFlow/IPFIX analysis.

2.3.2 Active Measurement

Active measurement is measuring the network by sending probe packets to monitor the statistics regarding these probe packets or the response to them to determine network characteristics [18]. Active measurement is used when the most up-to-date information about the behavior a network is needed. It provides great flexibility about the measurement metrics since usually, no capturing needs to be done ahead of time. It is possible to generate the desired type of traffic to measure, therefore, we usually do not have to process a large amount of data compared to the passive measurement approach. Note that the network traffic measured by this approach is not the real network traffic, but synthetically generated for specific purposes.

In active measurements, a specific set of packets are generated with the desired headers and payloads, then a set of target hosts are specified to send these packets to. In order to perform an Internet-wide active measurement, i.e., targeting the maximum amount of Internet hosts possible, there are two main challenges:

1. **Speed and rate limiting.** Internet-wide active measurements require scanning a large number of targets relatively fast to be able to provide timely measurements. The measurements also need to be configurable in terms of the scanning rate. There are multiple open-source tools that provide fast network scanning and configurability.
   - ZMap: provides stateless single packet scanning for the whole IPv4 address space [19] extended with IPv6 support [20].
   - ZGrab: provides stateful scanning most specifically useful when full handshakes are required [21].
   - Yarrp: provides stateless network topology discovery, similar to the traditional traceroute [22].
2. **Coverage**. Maximizing the number of Internet hosts we plan to target increases the coverage of the measurement, making the results more comprehensive and therefore, valuable. To be able to maximize the coverage, it's practical to distinguish between IPv4 and IPv6 hosts. The whole IPv4 address space includes 2^{32}, i.e., roughly 4.3 billion hosts, which can be targeted automatically using the above-mentioned tools. However, the IPv6 address space includes 2^{128} hosts making it impossible for the scanning tools to target the whole IPv6 address space.
   To overcome this, researchers have put together many different data sources containing active IPv6 addresses, including the IP addresses from different DNS datasets and RIPE Atlas traceroutes, resulting in an IPv6 hitlist [23]. Throughout this thesis, I have used this hitlist for all of the IPv6 measurements.

Throughout this thesis, the author uses active measurements with the help of the above-mentioned tools and mechanisms to analyze the behavior of Internet hosts to different types of traffic probes.

2.3.2.1 Ethical Considerations

Prior to carrying out active measurements, we go through an internal multi-party approval process which follows proposals by Partridge and Allman [24] and Dittrich et al. [25]. We adhere to scanning best practices [26] by restricting our probing rate, maintaining a blocklist, and utilizing dedicated servers that provide clear rDNS names, websites, and contacts for addressing abuse. These best current practices are followed both in Chapter 5 and Chapter 6. During our active measurements for Chapter 5, we received one email asking to be blocked, to which we immediately complied.

2.4 Chapter Summary

This chapter briefly overviewed the basic concepts making the Internet work, including the protocols and the modern Internet structure. I also discussed the importance of Internet measurements, different measurement approaches, and common formats and tools useful for Internet measurements.

3

Flowyager: Exploring Network-Wide Flow Capture Data

In the pursuit of uncovering hidden characteristics of Internet traffic, this chapter delves into the realm of analyzing network-wide flow data. In Internet organizations such as ISPs and IXPs, the network operators have to continuously keep track of the traffic over both long and short time windows. Over long time windows, e.g., days or hours, network operators are interested in provisioning network capacity or making informed peering decisions. Over short time windows, e.g., minutes, network operators would like to identify and rectify unusual events, e.g. attacks or network disruptions. To that end, they typically rely on either flow-level or packet-level captures from routers within their network [27]. For a summary of tasks and how previous work tackled them see Table 3.1.

In Section 2.3.1, we introduced the 5 features included in flow captures: source (src) and destination (dst) IP addresses, port numbers, protocol ID–to summarize traffic information per flow–Packet and byte count [28, 29]. Packet captures gather packet headers [30–33]. Unfortunately, gathering data for every packet is often too expensive at high-speed links. Thus, flow-level and packet-level capture tools rely on sampling packets, e.g., 1 of every 10k packets [34].

Recently, query-driven solutions, e.g., Sonata [35], Stroboscope [36], and Marple [37], made it possible to compile specific queries into telemetry programs and collect data from all queried network nodes. These solutions provide exceptional flexibility, but they require the network operator to know *a priori* (i) the nature of the network problem, (ii) the network-related query that has to be compiled into telemetry programs, (iii) the network node where the telemetry capability is available, and (iv) the node where the query has to be executed. Unfortunately, in large networks with hundreds of interfaces, operational issues arise at different parts of the network and the queries that are required are not known in advance. In many cases, network engineers have to try different queries to locate the source and type of problem interactively. Thus, it takes a prohibitively large time to compile such queries into telemetry programs. Another obstacle toward adopting such solutions is that this requires hardware investments by the network operator. For example, Marple relies on P4-programmable software switches that are not yet widely adopted by Internet Exchange Points (IXP) operators and Internet Service Providers (ISP).

To the best of our knowledge, there is at this point in time no system that offers answers to *a priori unknown network-wide* queries in a scalable *interactive* manner, even though the necessary raw network data, e.g., via NetFlow or IPFIX is collected by most operators.

From an operational point of view, fast exploration of large volumes of network flows over time and across sites is useful to answer a range of operational queries (see Table 3.1). Yet, network operators need to be able to tackle such tasks in a unified and systematic way with reliable and scalable tools. Existing data analytics systems, e.g., Spark [38], are not tailored to analyze network data when it comes to scalability, interactivity, handling of geo-distributed data, or answering a priori unknown network-wide queries.

In this chapter, we design, implement and evaluate a system, Flowyager, that is able to answer *a priori unknown network-wide queries* with fast response, and, thus, enables *interactive* exploration of network data *across network sites* and *over time*. The architecture of our system is built around the following requirements:

(1) Scalability: The system should grow with the network size, the number of data sources, and the analysis requirements. Hereby, it should enable distributed deployment and not require all data to be transferred to a central location.
(2) Reuse of existing flow captures: As it takes significant effort to deploy novel network capture utilities, the system should work on top of existing, widely deployed, and supported flow capture capabilities, such as NetFlow, sFlow, IPFIX, or libpcap. In high-speed links, these tools typically sample packets [34] to provide summaries of flow activity.
(3) Support of interactive and ad-hoc queries: To easily explore network data, the system needs to offer an interface that is flexible and interactive (meaning response times in the order of seconds) so as to improve user productivity and enable drill-down capabilities. Possible queries vary and a system should not only focus on batch-style known queries but also enable quick *ad-hoc* exploration of the data, i.e., answer queries that *are not known in advance*, and allow for follow-up queries. Answering network-wide queries should not require custom code or scripting as network operators usually neither have the required time nor the resources (e.g., storage or computing). The goal is to reduce the response time of queries from hours or dozens of minutes to seconds and, thus, enable interactive and drill-down queries.
(4) Support of queries across network sites and over time: Most queries are not just for some specific time period or network site. Rather, they correlate data spanning multiple periods, across network sites, and at different granularities, e.g., per site, region, time of day, and event. The system should be able to collect, index, and store summary data across multiple sites and over time.

Although most networks gather raw flow data, answering network-wide queries is difficult due to: (a) the distributed nature of data collection (per interface and router) at different locations, i.e., at multiple border and/or backbone routers, (b) the massive and ever-increasing size of the flow data (despite sampling) incurring an excessive cost to store, transfer, and analyze flow data–indeed, it often has to be deleted after some

Application	Related Work
Aggregated flow statistics (range queries over IP/ports/time/location)	[35, 37, 39–43]
Counting traffic	[35, 39, 42–49] [36, 50–56]
Traffic matrix	[46, 50]
DDoS diagnosis	[35, 44, 46, 50, 54, 57–59]
Super-spreaders Detection	[35, 44, 54]
top-K number of flows	[45, 55, 56, 60]
Flows above threshold T (Heavy Hitters)	[35, 41–45, 52–54, 61, 62]
Heavy Changers Detection	[44, 53, 54, 61–63]
Blackhole Detection	[46, 64–66]
Port-based / 4/5-tuple queries	[35, 39, 44–46, 51–53]

Table 3.1: Typical network queries and systems to tackle them. Currently, no system addresses all of them.

time to be able to store more recent data, and (c) the international footprint with the requirement to comply with local legislation which may prohibit the transfer of raw data.

To achieve the above, we need data structures that generate *succinct* and *space-efficient summaries*, as well as *indexing* of network flow captures that are light (easy to transfer), can be analyzed locally, and enable answering *interactive a priori unknown network-wide queries*. These data structures should be used to *accurately* and *quickly* answer queries and tackle network management tasks that involve *multiple sites* and/or span *multiple periods* in a *user-friendly* and *unified* way.

The contributions of this chapter are:

- We design, deploy and evaluate Flowyager, a system built on top of existing voluminous network captures, that enables interactive data exploration. We show that with Flowyager the query response time for network-wide queries can be reduced from hours or minutes to seconds.
- We propose a lightweight self-adjusting data structure, Flowtree, that inherits the performance of previously proposed hierarchical heavy hitter structures for computing flow summaries. Flowtree summarizes elephants as well as mice flows and supports multiple operators, such as merge, compress, and diff, to summarize information across multiple sites and time periods.
- We propose an SQL-inspired language, FlowQL, which provides a unified interface to ask arbitrary ad-hoc queries about flow captures, including drill-down queries.
- We show that when answering a wide range of queries, Flowyager significantly outperforms the state of the art data analytics systems, namely, ClickHouse, and Spark.

- We share our experience of rolling out Flowyager at different operational environments, namely a large IXP and a tier-1 ISP, and showcase how to tackle various network management tasks. We make Flowyager and its code available for non-commercial use [67].

3.1 Related Works

Existing network analytics systems, such as [68, 69], typically transfer the raw traces to a centralized data warehouse for archiving and processing. However, transferring the raw traces is increasingly expensive due to the data volume —e.g., Terabytes of flow data generated in a single day can be out of sync, and all need to be transferred. Moreover, additional constraints are posed by national regulations when networks operate at regions under different jurisdictions: for example, transferring data that includes user identifiers, e.g., IP addresses allocated to EU citizens, without their consent, violates the EU General Data Protection Regulation (GDPR) [70]. Fines are steep, namely up to 4% of worldwide turnover or 20 million Euros, whichever is higher.

Network monitoring systems: Alternative proposals suggest to enable powerful custom data collection per query and realize this by combining traffic mirroring and deterministic packet sampling. These include query-based monitoring such as Stroboscope [36], network troubleshooting using mirroring [71, 72], analysis of in-network packet traces [46, 73], as well as monitoring links on-demand as shown by Gigascope [39], pruning-based solutions such as Cheetah [74] or other SDN-based monitoring, such as [75] or PRECISION [76]. The main disadvantage of these systems is that the target flows, sites, and periods of interest need to be known in advance, which is often not the case in practice.

Streaming network telemetry systems, ranging from more classic approaches such as A-GAP [56] to the numerous modern solutions, such as Sonata [35], FlowBlaze [77] or Poseidon [78], build on the same ideas but require programmability from network devices, e.g., P4 switches or FPGA. These systems assume that users can predefine what is relevant and optimize the monitoring accordingly, often following a top-down approach [79]. As a consequence, if, potentially, all flows are of interest, these systems can degrade to "standard" flow monitoring which for large networks is challenging. Marple [37] adds flexibility to network-wide monitoring but requires P4-programmable capabilities that have not been yet widely adopted in wide-area networks by ISP and IXP operators.

Big data analytics systems: Some operators directly feed their flow captures into state-of-the-art analytics systems, often based on the map-reduce principle, e.g., Spark [38] and Hadoop [80], or column-based databases, e.g., ClickHouse [81]. This has scalability issues. Thus, recently proposed big data analytic systems–see [82–86] as well as [87] and references within–suggest to use a distributed setup whereby data is locally preprocessed, e.g., by aggregation or sampling, and then centrally analyzed. This reduces the need to transfer the raw data. Note that none of the above focuses

on network management tasks. Thus, their programming interface follows the map and reduce paradigm which differs from network operation tasks. Even though such systems can provide significant speedup for tasks that can be parallelized, not all network management tasks may benefit. Like Flowyager, such big data analytics systems are *flexible* w.r.t. the queries supported. Yet, unlike Flowyager, they typically are not *compatible* with existing network monitoring software, do not fully support principled *aggregation* (over time, space and flows), do not offer any *history*, and do not give any performance (accuracy or runtime) *guarantees*.

Data summaries–Heavy Hitters: Previous work on computing network summaries has focused on how to efficiently compute heavy hitters (HH) [30, 31, 88, 89] and hierarchical heavy hitters (HHH) [42, 90, 91] using minimal resources to be able to compute them on the router itself. These solutions provide an online summary of the (hierarchical) heavy hitters for a fixed observation window, at one location, and only on a given subset of the data. In contrast, to answer interactive network management queries (see Table 3.1), we need summaries over different subsets of the data, per site/router and across sites/routers, and at many different time granularities, from minutes to days — or even months.

Heavy hitters change as data is aggregated: as more data comes in, popularities increase overall. Consequently, the threshold to be considered a heavy hitter should be raised. In contrast, some HHH data structures, e.g., [42] use a single manually defined absolute threshold (e.g., frequency above 1000) to characterize heavy hitters, resulting in a data structure unable to adapt its definition of heavy hitter as the underlying data changes. Flowyager builds upon heavy hitter data structures by adding support for *aggregation* (over time, location, and flows) and adding *flexibility* w.r.t. the supported queries.

Data summaries–Sketches: Another approach for computing network summaries are sketches, e.g., [53, 92, 93] as well as systems that utilize sketches for network monitoring and debugging [44, 54, 94–96]. The capabilities of sketches include counting, top-K, HH, as well as HHH. They are highly space-efficient data structures that support many types of queries. Yet, most do not support range queries, e.g., queries that involve a range of sites and/or time periods. Moreover, extracting an estimate from sketches is often not time-efficient. We note that the focus of sketches is similar to that of HHH, i.e., computing online summaries for a fixed observation window with minimal resources. Flowyager could be built upon sketches but we decided to build upon a HHH data structure.

3.2 Flowyager Architecture

To address the challenges outlined earlier in this chapter, we build a scalable distributed network data analysis architecture, Flowyager. Its *input* is existing per-interface network *flow captures*, either flow summaries–reporting on packet, byte, or flow counts per 5-tuple (src/dst IP address, src/dst port, protocol)—or packet-level summaries (e.g., trace sample). We emphasize that we do *not* propose yet another

	Net. Mon.	Analyt-ics	HHH	Sketch	Flowyager
Input: Packets	✓	✗	✗	✗	✓
Input: Flows	✓	✗	✗	✗	✓
Distributed Queries	✓	✗	✗	✗	✓
Online	✗	✓	✗	✗	✓
Arbitrary Queries	✗	✓	✗	✗	✓
Query language	✗	✓	✗	✗	✓
Summarization	✓	✓	✓	✓	✓
Low Installation Cost	✓	✗	✓	✓	✓
Low Maintenance Cost	✓	✗	✓	✓	✓
Adaptivity to Data	✓	✗	✓	✓	✓

Table 3.2: Comparison of systems w.r.t. functionality offered. ✓: full support, ✗: no support

NetFlow. Its *output* is network reports including packet, byte, or flow counts across network sites and time periods. Prime users, i.e., network operators, can access the data via FlowQL, an SQL-inspired query language that returns results in seconds and, thus, enables interactive ad-hoc queries with drill-down capabilities. For a comparison between Flowyager and other approaches, we refer to Table 3.2.

To underline Flowyager's capabilities for exploring network data, we show in Fig 3.1 and Fig. 3.2 screenshots of Flowyager's Web interface. The Web interface highlights that searches are possible across time ranges, site sets, and feature sets. Moreover, it showcases Flowyager's drill-down capabilities that are also visually supported.

Flowyager is a modular system that consists of three main components:

1. *FlowAGG*, which takes existing flow (or packet) captures as input and computes flow summaries, using *Flowtrees* (see below), which it stores and exports. Besides, *FlowAGG* may, if it has enough storage, keep a local copy of the flow captures themselves.
2. *FlowDB*, which takes flow summaries as input, stores, and indexes them, while using them to answer FlowQL queries. It can use FlowAGG internally to compute further flow summaries.
3. *FlowQL*, which uses the flow summaries kept within FlowDB to answer interactive or batch-style queries including Hierarchical Heavy Hitter/top-K queries, Above-Thresh queries, or top-K heavy changer queries across time and sites.

To better understand the system architecture, Figure 3.3 gives an overview of the overall system, while Figure 3.4 presents Flowyager's processing pipeline. Each router sends its data to a NetFlow collector (1), which forwards it to one of potentially many distributed FlowAGG instances (2). Each FlowAGG instance computes summaries (3) and then uploads these either to another FlowAGG instance or directly to

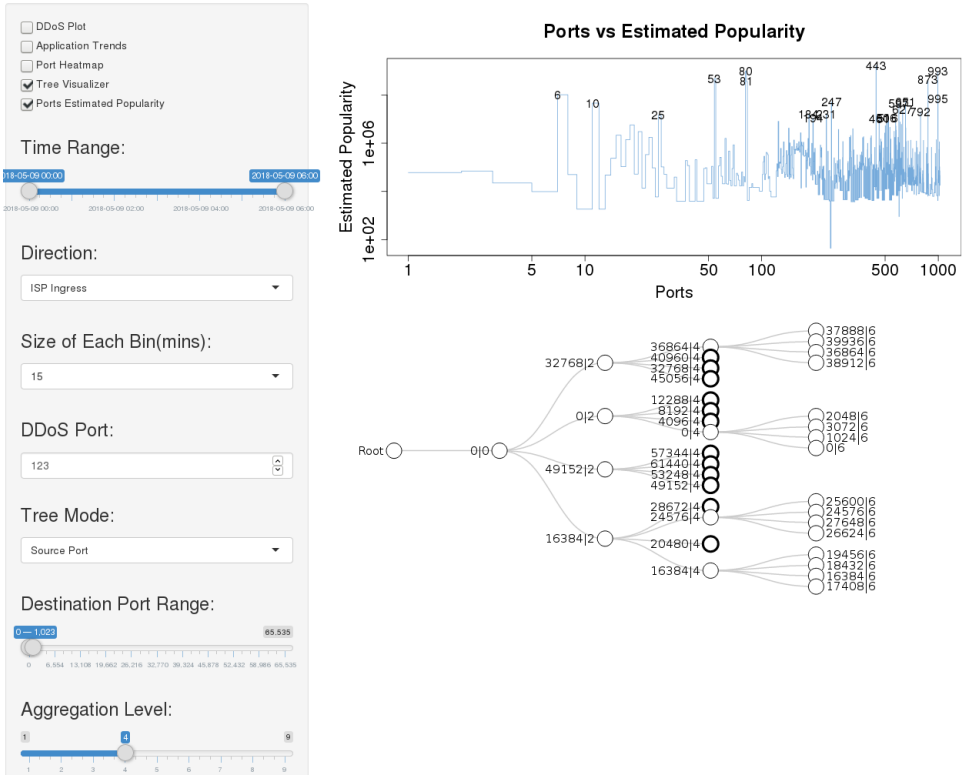


Figure 3.1: Flowyager: Interacting with 1-feature Flowtrees

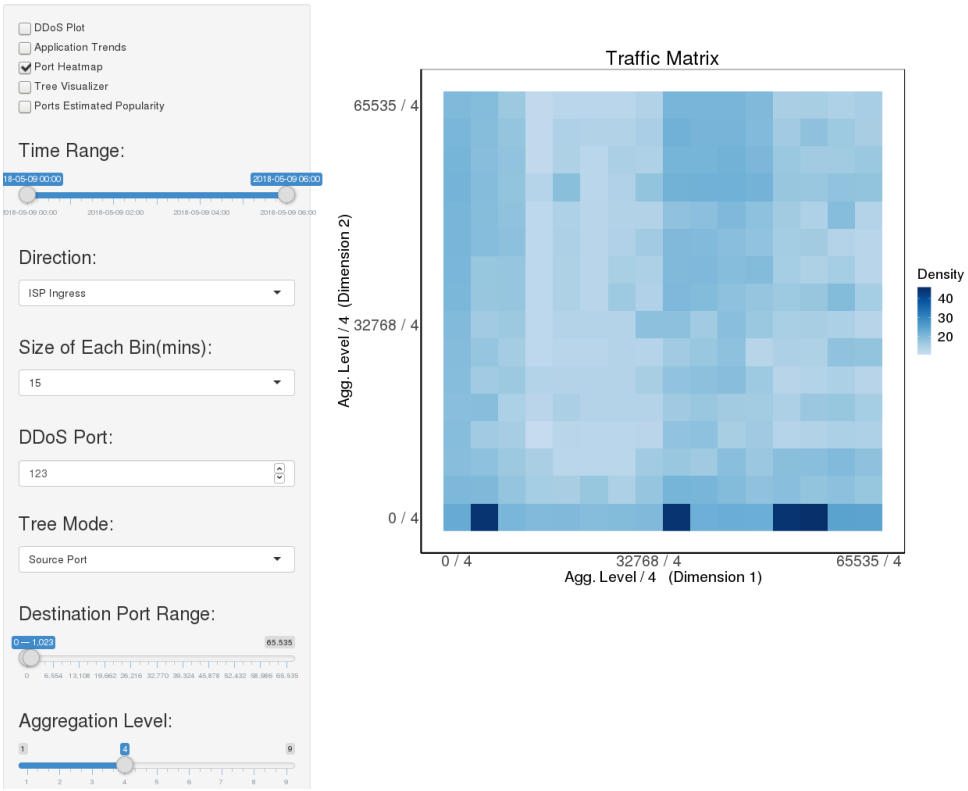


Figure 3.2: Flowyager: Interacting with 2-feature Flowtrees

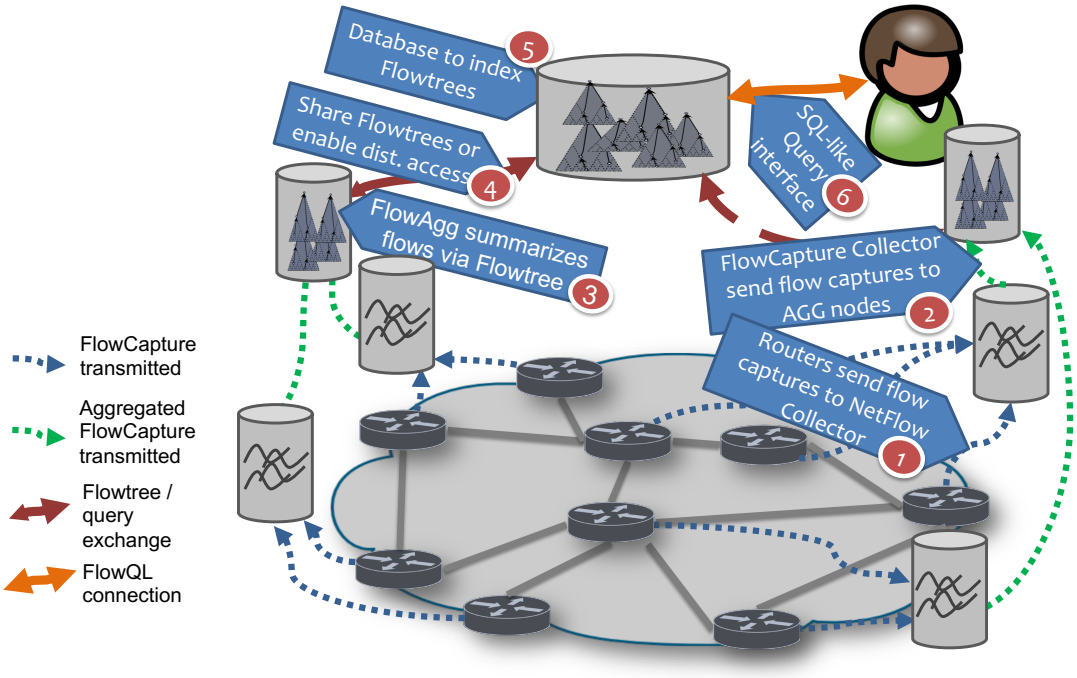


Figure 3.3: Flowyager architecture

FlowDB (4)[1]. FlowDB then processes the summaries (5) and uses them to answer user queries (6).

Flowtree is a data summary of a stream of raw flow data that supports efficient 1-d HHH extraction and other operators. Flowtrees are the data primitives of Flowyager. Details on the design and implementation of Flowtree data structure and Flowtree operators are presented in Section 3.3.

FlowAGG uses a separate plug-in, written in C, for each data source, including IPFIX, NetFlow, sFlow, and libpcap.

FlowDB is responsible for collecting and storing the Flowtrees. It also provides an interface that the user of the Flowyager can use to answer network-wide queries based on the stored Flowtrees, **FlowQL**, whose design is largely inspired by GSQL [39] which uses an SQL-like query language. Using GSQL directly does not suffice due to the unique capabilities of Flowyager. Details on the design and implementation of FlowDB are presented in Section 3.4.

In total, it took approximately 21k lines of code (LoC) in C and C++ to realize Flowyager. About 16k LoCs are for FlowDB, 1.5k for FlowAGG, 2.5k for Flowtree library, and 1k for shared components.

[1]For simplicity we restrict our discussion to a centralized instance of FlowDB. However, it is possible to use a hierarchical design similar to what has been proposed for logs of distributed servers [97, 98]

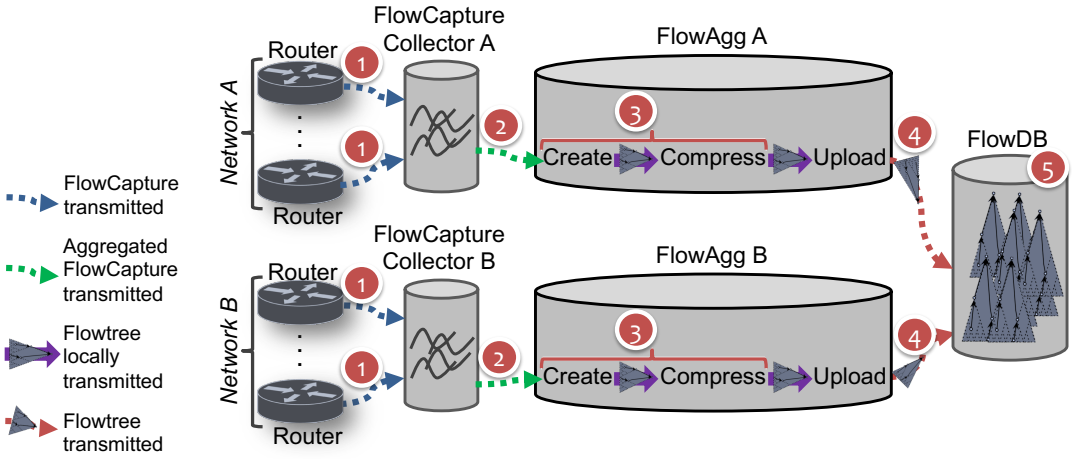


Figure 3.4: Flowyager processing pipeline

3.3 Flowtree

Flowtree is the data structure that is used as a data primitive in Flowyager. Before we dive into the details of Flowtree and its operators, we provide background on Hierarchical Heavy Hitter (HHH) data structures.

3.3.1 Hierarchical Heavy Hitters

To enable Flowyager we need succinct summaries from flow captures that are light to transfer, yet, allow for real-time, interactive queries using different flow feature sets. A *flow feature* refers to any of the components of a flow's 5-tuples, namely protocol, src and dst IP, src and dst port. A *feature set* includes a subset of the possible 5 flow features.

We take advantage of the fact that most of the data on the Internet is skewed in the sense that Zipf's law [99–101] typically applies. However, flat summaries, i.e., histograms, do not suffice. Rather, we need hierarchical heavy hitters (HHH) [2]. HHH utilize attribute hierarchies and identify the most popular elements across a hierarchy. For IPv4 prefixes, we use the network prefix length as an obvious feature hierarchy. As such, 10.1.2.0/23 is the parent of 10.1.2.0/24 and 10.1.3.0/24. For ports, we can use port ranges, e.g., 80/15 is the parent of 80/16 and 81/16. Each feature hierarchy, by default, uses a mask. An IP a.b.c.d is part of the prefix a.b.c.d$|n_1$ and a.b.c.d$|n_1$ is a more specific prefix and, thus, a child of a.b.c.d$|n_2$ if $n_1 > n_2$. The same applies to ports, whereby, e.g., 0|8 refers to the ports from $[0, 63]$. It is possible to define custom hierarchies, e.g., all Web ports, all DNS ports, or all well-known ports.

[2] The set of HHH for a single hierarchical attribute with popularity counts and a threshold θ corresponds to finding all nodes in the hierarchy such that their HHH count exceeds $\theta * N$, whereby the HHH count is the sum of all descendant nodes which have no HHH ancestors.

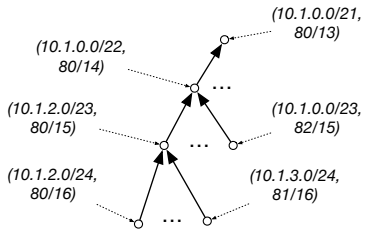


Figure 3.5: Example: 2-Feature flow hierarchy

Ideally, one would use 5-dimensional hierarchical heavy hitters (5-d HHH), across all flow features. Unfortunately, this is infeasible due to its computational complexity [90, 102]. Rather, we use 1-d HHH which can be updated in amortized $O(1)$ time per entry while maintaining the accuracy for HHH and space efficiency of $O(H/\epsilon \log(\epsilon N))$, whereby N is the number of items processed, H is the number of hierarchy levels, and ϵ bounds the precision [90, 102].

Contrary to previous work, we do not restrict the 1-d HHH to a single flow feature. Our first key functionality is that we can generalize 1-d HHH by defining a *joined hierarchy* for a given feature set, e.g., a joined hierarchy for both dst IP and dst port, whereby, the parent of 10.1.2.0/24|80/16, as well as 10.1.3.0/24|81/16 (IP range|port range) is 10.1.2.0/23|80/15. The parent of 10.1.2.0/23|80/15 is 10.1.0.0/22|80/14 and its great-grandparent is 10.1.0.0/21|80/13. For visualization of a sample 2-f hierarchy see Figure 3.5. In effect, we rely on *generalized flows*: Flows summarize related packets over time at a specific aggregation level. Possible feature sets include "4-feature" flows (i.e., (src IP, dst IP, src port, dst port)), "2-feature" flows, e.g., (dst IP, dst port) (DIDP).

The joined hierarchy can capture the correlation of more than one dimension, e.g., the correlation between IP activity and port activity. It allows identifying heavy hitters on sets of features, and thus, investigating more complex use cases. For example, in an attack, both the target IP and port are important to investigate the type of attack. In general, any query that involves multiple features can be potentially benefited by this joined hierarchy.

Our second key functionality is that if the 1-d HHH data structure supports the operators *merge* ($\cup$) and *compress*, we can compute summaries across time and/or space. In effect, these two operators allow us to add the features *time* and *location*. Given two data structures, A_1 for time period t_1 (location l_1) and A_2 for t_2 (l_2), we get the joined data structure by $A_{12} = (A_1 \cup A_2)$. The *compress* operator is especially useful in reducing the memory footprint of the structure. This operator prunes the tree leaves, and if needed the internal nodes, whose contributions are less than some configurable thresholds, and summarizes their contribution to their parents.

Other operators are *diff*, *query*, *drill-down*, *HHH* resp. *TOP-k*, *Above-x* The diff operator is useful to identify changes, the drill-down operator to explore sub-regions. The HHH and Above-x operators allow us to find popular feature sets. The operators are used for interactive queries via FlowQL.

Algorithm 1 Flowtree: Creation/ update

```
Function: Build_Flowtree (pkts resp. flows)
1: Initialize Flowtree
2: for all pkts/flows do
3:   Extract_features(pkt resp. flow).
4:   Construct node from features.
5:   Add (Flowtree, node, feature set).
Function: Add (Flowtree, node, features)
1: Add_node(Flowtree, node, features).
2: next = next_parent(node).
3: while next != parent(node) or (next ∈ tree). do
4:   Add_node(Flowtree, next, NULL) with probability p.
5:   next = next_parent(next).
Function: Add_node(Flowtree, node, features)
1: if node exists then
2:   comp_pop[node] += stats(flow/pkt).
3: else
4:   Insert node with comp_pop[node] = stats(flow/pkt).
5:   parent(node) = find_parent(Flowtree, node).
6:   for child in children(parent(node)) do
7:     if child ∈ node then
8:       parent(child) = node.
```

Algorithm 2 Flowtree: Stats and Compress operator

```
Function: Stats(Flowtree)
1: Initialize pop to comp_pop for all nodes
2: Node_list = nodes of Flowtree in DFS order
3: for node in Node_list do
4:   pop[parent(node)] += pop[node]
Function: Delete(Flowtree, node)
1: parent = find_parent(Flowtree, node).
2: comp_pop[parent] += comp_pop[node].
3: children(parent) += children(node).
4: Free node
Function: Compress(Flowtree,
   thresh_comp_pop, thresh_pop)
1: Stats(Flowtree).
2: for each node do
3:   if (node is leaf and
     comp_pop[node] < thresh_comp_pop) then
4:     Delete(Flowtree, node)
5:   else if (comp_pop[node] < thresh_comp_pop
     and pop[node] < thresh_pop) then
6:     Delete(Flowtree, node)
```

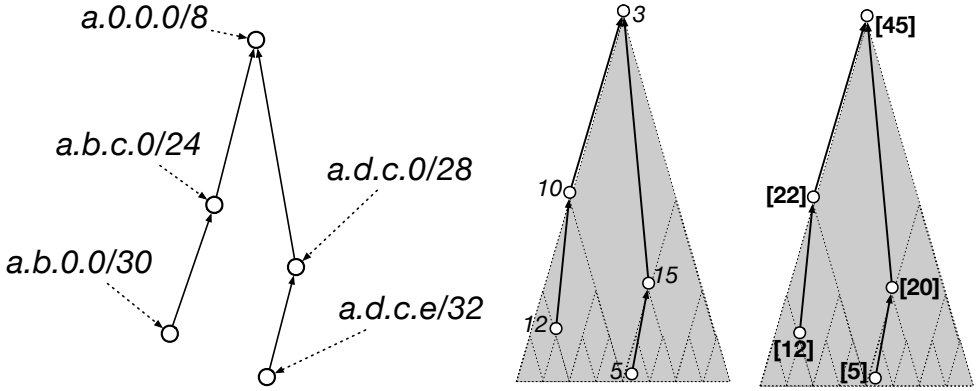
Algorithm 3 Flowtree: Operators

```
Function: Merge(Flowtree 1, Flowtree 2)
1: Flowtree = Flowtree 1
2: for each node in Flowtree 2 do
3:   Add_node(Flowtree 1, node)
Function: Diff(Flowtree 1, Flowtree 2)
1: Flowtree = Merge(Flowtree 1,Flowtree 2)
2: for each node n in Flowtree 2 do
3:   comp_pop(n) = abs(comp_pop(n) -
     2*comp_pop2(n)).
```

3.3.2 Flowtree Data Structure

After evaluating different 1-d HHH data structures, including those of Cormode et al. [90, 102], Basat et al. [42], and Mitzenmacher et al. [91], we decided to augment the structure by Cormode et al.: this data structure is self-adjusting and its entries can be easily extracted via enumeration; thus, it provides natively drill-down capabilities. Flowyager does *not* intrinsically depend on this data structure; rather, it can be built on top of any data structure that supports abstract hierarchies and the basic operators.

Flowtree data structure: Generalized flows form a tree via its hierarchy where each node corresponds to a flow. An edge exists between any two nodes a, b if a is a subnode of b in the feature hierarchy, i.e., if $a \subset b$ —see Figures 3.7(a) and 3.7(b). We annotate each node with its popularities, including packet count, flow count, and byte count for UDP and TCP. The popularity of a node is the sum of its own popularity and the popularity of the children–see Figure 3.6(c).



(a) 1-feature Flowtree: IP. (b) Comp_pop (c) Popularity

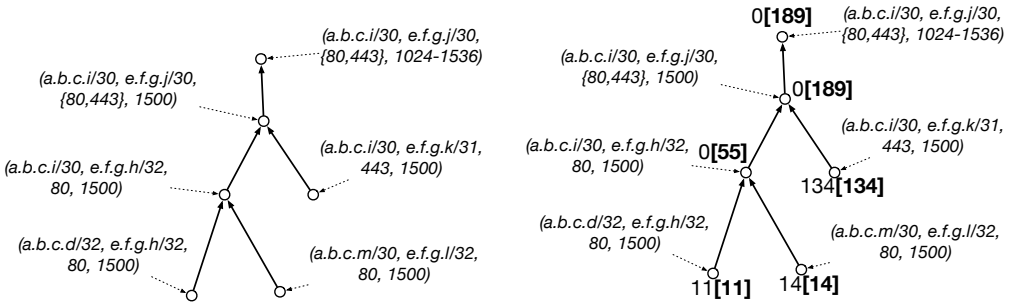
Figure 3.6: Flowtree concept.

However, during the construction of the trees, we only keep the nodes' "complementary popularity", namely the popularity (pop) that is not covered by any of the children. Thus, it is possible to prune such a tree by pushing the contribution of the pruned nodes to their parent. This is a *key functionality* for efficiently updating our self-adjusting data structure. Flowtree keeps "popular" nodes and prunes "unpopular" ones by summarizing them at their parent. Flowtree inherits the insertion and self-adjusting strategy from Cormode et al. but rather than allowing the number of nodes to grow unlimited, we limit the maximum number of nodes that a tree can contain by repeatedly pruning (compressing) the tree when necessary. Still, Flowtree closely matches the excellent performance and accuracy bounds for 1-d HHH in terms of space efficiency and precision.

3.3.3 Flowtree: Visualizing the Concepts

We start with the visualization of the differences between popularities and complementary popularities in Figure 3.7. Next, we show the two different feature hierarchies, namely a 1-feature hierarchy on IP addresses, and a 4-feature hierarchy on src/dst IP addresses and src/dst ports with and without popularities, see Figures 3.6(a) and 3.7.

Initially, a Flowtree has exactly one entry–the root. When adding a node, we add a new leaf node if necessary and a subset of the nodes on the path to the first existing parent, (in the worst case the root) and update the statistics of the leaf node. We call these intermediate nodes as internal nodes. Thus, each node maintains the complementary popularity (comp_pop), the popularity (pop) that is not covered by any of the children, see Alg. 1. Popularities are computed from the complementary popularities by summing the complementary popularities of all nodes in its subtree including its own. This can be done via a depth first search in O(# nodes) time, see Alg. 2. This uses two functions for finding parents of a node. parent(node) refers



(a) 4-feature Flowtree:src/dst IP and port.

(b) 4-feature Flowtree with complementary_pop. and pop.

Figure 3.7: 4-feature Flowtree.

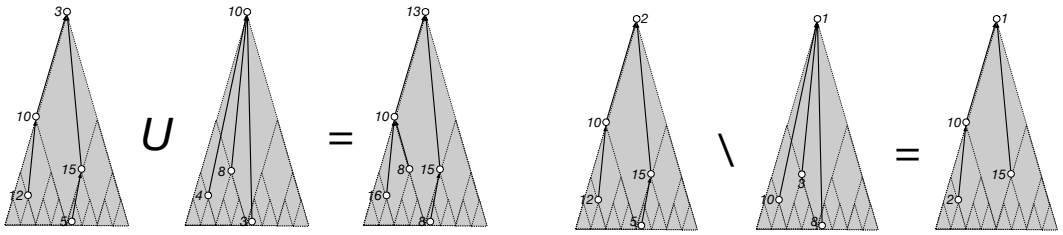
to the direct parent in the feature hierarchy while find_parent(node) refers to the parent in the Flowtree.

Updating an existing node corresponds to finding it, which takes time O(1) using an appropriate hash-map. Adding a new node may take up to O(# hierarchy level) time (using an appropriate hash-map). Yet, the expected number of new nodes is small if the distribution of the data is skewed.

To limit Flowtree memory footprint, we periodically or on demand, delete nodes with low popularity. We first compute the popularities by using the stats function in Alg. 2 and then prune nodes whose complementary resp. absolute popularity are below an adjustable threshold. This ensures that at any time the number of nodes in a Flowtree is proportional to the number of processed flows resp. less than a predefined maximum. The complementary popularity of a deleted node as well as its children are pushed to its parent. The overall cost of such a compression step is O(# nodes). Note that since only nodes with small popularity are deleted, the complementary popularity of an interior node is a good estimate of the cardinality of the contributing flow set. Finally, to control the rate of the growth of the tree and preventing the frequent addition and deletion of internal nodes, we insert the internal nodes with a probability of p. The default value of p is 0.3.

3.3.4 Flowyager Operators

Query and drill-down: The base operators are *query* and *drill-down*. If the feature f is a node in the Flowtree, the answer is computed from the node statistics. Otherwise, we find the potential node, q, that corresponds to f and estimate its popularity based on the popularity of the predecessor of q, p, and its children, C. We split the children into two subsets: C_f and $C_o = C - C_f$, whereby C_f includes those that are a subset of f in the hierarchy. Now, $\sum_{c \in C_f} \text{pop}(c)$ is a lower bound for the popularity of f and two estimates of f's popularity are $\text{pop}(p) - \sum_{c \in C_o} \text{pop(c)}$ or $\text{comp_pop}(p) + \sum_{c \in C_f} \text{pop}(c)$i.If the feature set does not correspond to a node p, the query is expanded to a tree-walk starting at the smallest possible parent of p. The output of the query are then all nodes and their popularities that match the input



(a) Merge (b) Diff

Figure 3.8: Flowtree Operators: Merge and Diff

feature set. For example, src_ip = a.b.0.0|16 and src_port = 80|16 start at node (a.b.0.0|16,80|8) and outputs only the nodes where src_port is 80 and src_ip is a subprefix of a.b/16. Drill-down queries retrieve the children of a node. Note that we can derive estimates for all flows, from mice to elephants: even for low-popularity nodes, the number of flows remains a good estimate for the number of contributing flows.

Above-t: Results in a tree-walk and all nodes whose popularity are above the threshold value are returned.

Top-k : To compute the top-k, we identify the Flowtree entry with the largest popularity, delete its contribution, and then iterate. Hereby, we use a priority queue.

Merge: We merge two Flowtrees by adding the nodes of one to the other. Note that the update will only be done for the complementary popularities— see Alg. 3 and Fig. 3.8(a)—, with missing nodes being assigned a popularity of zero. The statistics have to be recomputed and, to reduce the memory footprint, we compress the joined tree. If the total absolute contributions of the two trees differ significantly, one should rescale the complementary popularities of the trees before merging.

Diff and HeavyChanger: Just as one can merge Flowtrees, one can also compute the difference between two trees. This is a merge operation with subtraction instead of addition–see Alg. 3 and Fig. 3.8(b). Heavy changers are detected by using Top-k on diff of the two trees.

Flowtrees maintain counters for various features of the flows. In the current implementation, we use counters for packet, byte and flow counts. This structure supports cardinality-based queries but is limited to the elements (features) already in the tree (nodes). It is possible to maintain additional counters and support additional cardinality-based queries, e.g., using counters for ports, but at the cost of requiring additional space. In some cases, this is necessary. For example, such cardinality-based queries will enable the detection of non-volumetric attacks, e.g., semantic attacks. By allocating more space and maintaining more counters, it is possible to detect different types of attacks, e.g., “slow” DDoS attacks (Slowloris). We plan to explore the accuracy of cardinality based queries and the effect of allocating more space and maintaining more counters in Flowtrees as part of our future in future work.

3.4 FlowDB

FlowDB collects and stores Flowtree summaries computed by FlowAGG in persistent storage. Each Flowtree has a unique key that is made from its timestamp which along with its granularity reflects a time interval, the id of the site/location, and its feature-set. The values are the Flowtrees, which are stored as byte buffers. Figure 3.9 visualizes FlowDB's architecture.

3.4.1 FlowDB Implementation

Currently, our database of choice is MongoDB [103] because it is lightweight, although any other key-value datastore can be used. To accelerate query processing, we use an in-memory index and an in-memory cache. The in-memory index is a collection of T*-trees that track Flowtrees and enable range queries over different time periods. The in-memory cache uses a least recently used (LRU) policy to keep recently added or queried trees in memory. FlowDB is designed with parallelization in mind: it is capable of receiving multiple streams of Flowtrees from multiple FlowAGG daemons while answering queries to multiple users at the same time. Parallelization is employed in performing major tasks such as handling requests from FlowAGG daemons and remote API calls, storing Flowtrees in persistent storage, and query processing. Upon receiving a query, the system first checks whether the queried trees are in memory. In case of cache misses, it retrieves trees from storage.

The system is highly configurable in terms of memory usage, by setting a maximum number of Flowtrees in memory, cache eviction interval, degree of parallelization, etc. The maximum number of Flowtrees in memory controls the memory footprint of FlowDB. To access the database, FlowDB offers both an API with the services Add Flowtree and Get Flowtree and an interface for FlowQL. FlowAGG and other components of Flowyager use the Apache Thrift Remote Procedure Call (RPC) framework [104] for communication.

To enable *Geo-Distributed Query Execution*, the in-memory index keeps track of whether a Flowtree is stored locally or at a remote FlowDB. Thus, if necessary, all remote Flowtrees can be fetched via the FlowDB API to answer a FlowQL query. In our planned geo-distributed query execution, we partition site-IDs and map a site-ID to a FlowDB instance. Once a FlowDB instance receives a query, it will check whether the given site-ID is stored locally. If the required Flowtree is not stored locally, it can issue a request to the target FlowDB instance and retrieve the Flowtree. Once the Flowtree is retrieved, it will be merged with the Flowtrees that are already present and the intended query is fulfilled. The evaluation of this feature is beyond the scope of the current manuscript.

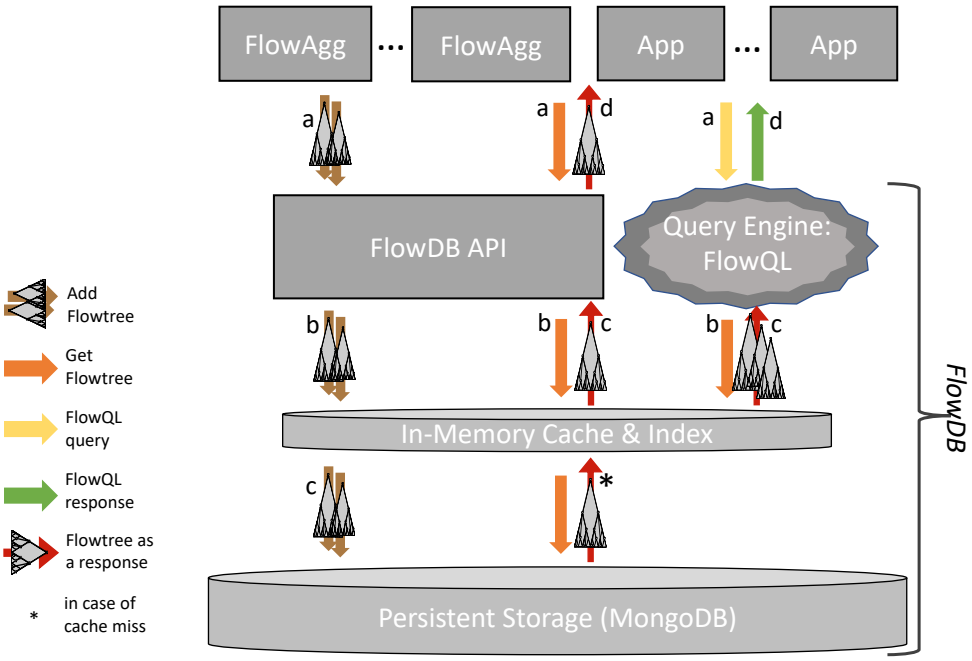


Figure 3.9: FlowDB overview

3.4.2 FlowQL Query Language

To realize FlowQL, we took inspiration from SQL keywords, yet we developed our own grammar. We used ANTLR [105] to generate the parser for the grammar. We offer an interactive command-line shell as well as a graphical user interface using R shiny [106]–cf. the screenshots from Fig 3.1 and Fig. 3.2. More specifically, with FlowQL the user chooses their operator via a `SELECT` clause, one or multiple time periods via a `FROM` clause, and the feature set via a `WHERE` clause.

SELECT: specifies the answer type. Allowed values include *'pop'* for popularity or flow/byte/packet count, *'top-K'* for the top-k most popular flows, *'HHH-P'* for the 1-d hierarchical heavy hitters with flow counts above P% of total traffic, *'hc-K'* for the top-k heavy changers, *'above-T'* for all flows with popularity above t, and *'*'* for all flows satisfying the WHERE clause.

FROM: specifies one or multiple time periods.

WHERE: selects the feature sets and one or multiple conditions. Possible feature elements are *site_id*, *src_ip*, *dst_ip*, *src_port*, *dst_port*, *proto*. Possible values are ANY or any region, IP prefix, or port range (using the IP|mask resp. the port|portmask syntax). Combinations are feasible via (AND, OR, and ()).

Thus, FlowQL queries have the following syntax:

```
SELECT [pop, top-k, hc-k, above-t, hhh-k, *]
FROM (time YYYY-MM-DD hh:mm to YYYY-MM-DD hh:mm)+
WHERE ([Conditions via AND, OR, ), (, feature = value])+
```

Using FlowQL, we found that we often wanted to repeat the same query across multiple time bins or sites. Thus, we added two iterators: `answer-bin-x` that iterates

across time bins of size x minutes and `site_id=ITR-x|n` that iterates across all sites within a site set, specified with an interval, e.g., $[x, x + 2^n - 1]$, or using a pattern.

To be able to drill-down and inspect a specific time-range in more detail, we additionally provide drill-down queries. In a drill-down query, a particular granularity in which one desires to inspect the traffic should be specified. For instance, to see the result of a query in 15-minute time bins, one should specify *bin15* in the query.

3.4.3 Query Execution

Upon receiving a FlowQL query, first, the WHERE clause is converted into a Disjunctive Normal Form. This results in breaking down the current query into smaller queries, which we call *mini-queries*.
Each mini-query is then processed independently. For each mini-query, the corresponding trees are fetched considering the time-range, granularity, and feature sets. For instance, for a query requiring *src_port=X*, 1-feature trees, *SP* in this case, are fetched. In a non-drill-down query, trees with the highest granularity existing in FlowDB are fetched. For a drill-down query, trees with the granularity specified in the query are fetched. If the specified granularity does not exist in FlowDB, multiple lower-granularity trees are merged using the MERGE operator to build trees with the specified granularity. Consider the following query which asks for bin-30:

```
 SELECT pop(any,byte,bin30) FROM (time 2018-05-09 00:00 to 2018-05-09 23:59)
WHERE site_id=ANY and src_port=X
```

This is a drill-down query to zoom into a full-day time-range in half-an-hour bins. Now assume that there are no 30-min granularity trees in FlowDB for the specified time-range, but there are 15-minute granularity trees. Then for each time-bin, two 15-minute trees will be merged to build the required granularity.
If the number of trees to be merged is large, the merge operation is performed in parallel to speed up the merge process. In a heavy changer query, two time-ranges should be provided and the trees fetched for each of these two time-ranges are diff'ed using the DIFF operator.
Then, the final trees are processed using different Flowtree operators to fulfill the query conditions, e.g., *src_port=X*. If the query is *pop*, knowing the popularity is as easy as finding the corresponding node in the tree and returning the popularity value. If the node is not in the tree, an estimation using the parent's popularity is returned as previously described in 3.3.4.
If the query is *above-T*, ABOVE-T operator with threshold T is used. For the *top-K* and *hhh-P*, the TOP-K operator will be used. In top-K, it should return the top K flows with any non-zero popularity. In hhh-P, P is the threshold for the fraction of total contributions.

Dataset	Time range	#Interface	Input Size	Type	Time bin
IXP-2019-09	Sep'19 1–7	≈ 1,250	≈ 10TB	Flow	15m
ISP-2019-04	Apr'19 1–2	≈ 1,300	≈ 25TB	Flow	15m
MAWI-2018-05	May'18 9–10	2	≈ 1TB	Packet	1m

Table 3.3: Deployment overview: IXP-2019-09, ISP-2019-04, and MAWI-2018-05

Short Form	Meaning
SIDI	src IP and dst IP
SPDP	src port and dst port
SISP	src IP and src port
SIDP	src IP and dst port
DISP	dst IP and src port
DIDP	dst IP and dst port
SI	src IP
DI	dst IP
SP	src port
DP	dst port
FULL	src IP, dst IP, src port, and dst port

Table 3.4: Overview of the feature sets of Flowtree

3.5 Experimental Deployments

We rolled out and tested Flowyager in three different types of networks, namely a large European IXP (IXP-2019-09), a tier-1 ISP (ISP-2019-04), and our testbed using a sample dataset (MAWI-2018-05)—see Table 3.3 for an overview. In this chapter, we report on experiments on stored data that we use for reproducibility. At two locations, the IXP and the ISP, we are in the process of moving towards live data import after extensive testing on site.

Ethical considerations: We are fully aware of the sensitivity of network data and, therefore, only work with a subset of the packet header information, namely src IP, dst IP, src port, dst port, protocol, whereby all IPs have been consistently anonymized per octet (bijective substitution using a hash function), even though this may negatively affect prefix aggregation. Note that the live operational deployment of Flowyager will not require such anonymization.

IXP-2019-09 Dataset: This dataset consists of IPFIX flow captures at one of the largest Internet Exchange Points (IXPs) in the world with more than 800 members and more than 8 Tbps peak traffic. The IPFIX flow captures are based on random sampling of 1 out of 10k packets that cross the IXP switching fabric. The anonymized capture includes information about the IP and transport layer headers, as well as packet and byte counts. To evaluate the system at real-world scales, we included all

sites during the first week of September 2019. Each site corresponds to the router interface of an IXP member connected to the IXP's switching fabric.

We deployed Flowyager within a virtual machine (VM) on a server at the IXP's premises. The VM is assigned 400 GB of memory and 40 threads on a machine with two Intel-Xeon-gold 6148 CPUs each with 40 threads.

ISP-2019-04 Dataset: This dataset consists of approx. 1,300 NetFlow streams (one per interface) from a major tier-1 ISP. We receive NetFlow data from 40 routers located in 30 cities in 4 European countries, as well as the US. The ISP's internal systems preprocess the raw NetFlow streams into 26 separate ASCII data streams. The NetFlow packet sampling is identical across all the routers. We include all data from Apr. 01, 2019 (00:01:00 UTC) to Apr. 03, 2019 (02:01:00 UTC). We deployed Flowyager as a Docker container with 94 GB memory and 32 threads on a machine with two Intel Xeon E5-2650 CPUs.

MAWI-2018-05 Dataset: This dataset consists of packet-level capture collected at the transit 1 Gbps link of the WIDE academic network to its upstream ISP on May 9-10, 2018. Each packet capture lasts for 15 mins and contains around 120 M packets. The anonymized trace is publicly available [107] and we use it to be able to release sample queries and results. We interpret each direction as a site. For this dataset, we deployed Flowyager on a testbed machine, with 128 AMD-EPYC 7601 CPUs and 1.5TB memory.

Flowyager setup: In terms of the basic setup for the Flowyager evaluation, we choose fixed time periods rather than a fixed number of flows. The advantage of the former is that we can easily summarize across time and that we can even look at coarser time granularities. The advantage of the latter is a constant number of entries to summarize. We choose the former rather than the latter as summarizing and investigating across time are typical network operator tasks. We keep Flowtrees for every 15 minutes for every site for the IXP-2019-09 and ISP-2019-04 datasets and 1 minute for the MAWI-2018-05 dataset. We generate 11 different feature trees, namely all four 1-feature trees, all six 2-feature trees, and a 4-feature tree, see Table 3.4 for the details. By default, we limit each Flowtree to 40k nodes. 1-feature port Flowtrees are limited to 10k nodes. In addition, we generate aggregated trees for 15 minutes, 1 hour, 1 day, and 1-week time granularities, each with at most 40k nodes. This results in one tree per site for each time granularity and a single tree for all sites for each time granularity.

Big data analytics setup: We compare Flowyager's performance with *task-specific data-parallel Python scripts*, as well as installations of a prominent big data analytics platform, namely *Spark* [38], and a column-based state of the art database, namely *ClickHouse* [81]. Each installation was done on the same VM as Flowyager. Note that this implies that Spark was not deployed on a physical cluster of machines but in a multi-threaded environment.

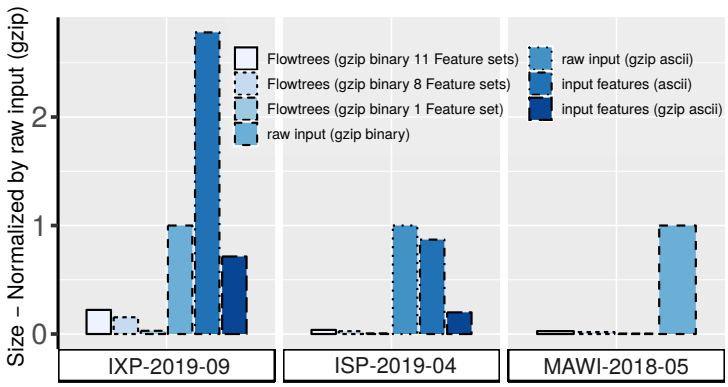


Figure 3.10: Space usage vs. raw compressed (gzip) input data

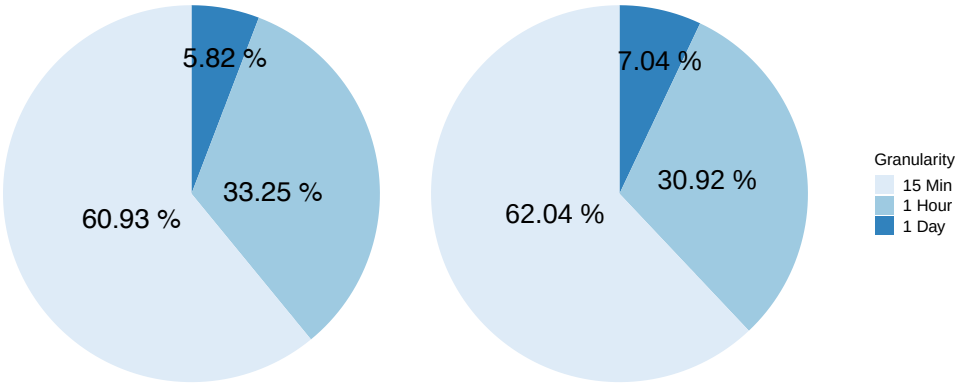
3.6 Flowyager Prototype Evaluation

Next, we describe our experience with deploying Flowyager, which we will make publicly available for non-commercial use. Our evaluation highlights the four main strengths of Flowyager: reduced storage footprint, low transfer cost, rapid response to a wide range of queries, and high accuracy. We start the evaluation with Flowyager's memory footprint, then we move on to evaluating FlowQL for query response time, and its CPU and disk I/O footprint.

3.6.1 Flowyager Evaluation

Flowyager space efficiency: For the IXP-2019-09 (ISP-2019-04), we see that compared to the original compressed IPFIX data (original compressed ASCII flow summaries), the single full-feature Flowtree in compressed binary format has a space saving of 97% resp. 99.5%. With additional feature sets, e.g., all 1-feature Flowtrees and three 2-feature Flowtrees, we still reach space saving of 92% resp. 97.5%. If we include all 11 possible feature combinations, the space saving is 89% resp. 96%. Even if we normalize not by the raw input data but only against the necessary features for the Flowtrees, the space savings are still excellent, e.g., more than 97% for the 1-feature Flowtree at the ISP-2019-04. For a visualization of the space efficiency relative to the size of the raw compressed (gzip) input data, see Figure 3.10(a).

While 15-minute time granularity is excellent for answering detailed queries, many queries involve coarser time granularities. Thus, it can be useful to add time as another feature and add 1-hour as well as 1-day aggregated Flowtrees by merging (and then compressing) the smaller-time-granularity Flowtrees. Flowyager does so automatically. While this needs some extra memory, it adds less than 40% overhead–see Figure 3.11— while offering the potential to significantly reduce query response time. Moreover, should space become an issue, Flowyager may decide to permanently delete smaller-time aggregates while keeping higher-time aggregation summaries. This is one of the design features that enable resource management with Flowyager. It is always



(a) IXP-2019-09 (b) ISP-2019-04

Figure 3.11: Pie Chart: MongoDB footprint.

possible to still keep coarse grain summaries of previous time periods or site sets even if disk space is running out.

3.6.2 FlowQL Evaluation

Next, we focus on the performance (query response time) of the query capabilities and the query engine using a set of benchmark queries. In particular, we go back to the main tasks of a network manager–recall Table 3.1—and pick a benchmark query for each of the identified tasks–note that the detection of one super-spreader requires two queries. These chosen queries are shown in Table 3.5, the table which thus contains queries for every single important network management task tackled by related work.

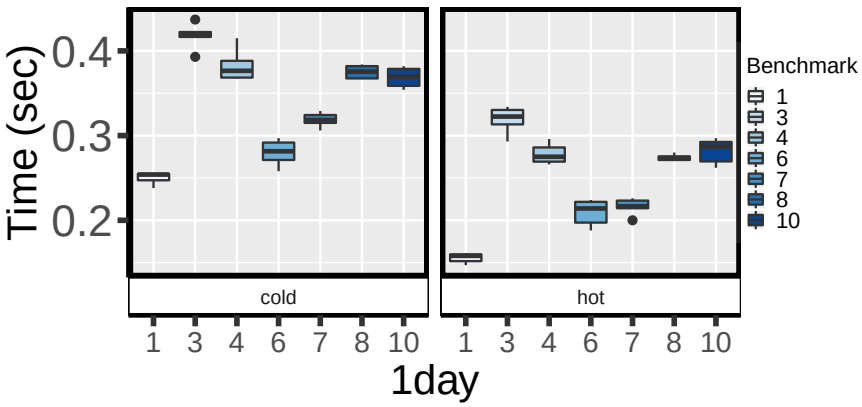
To challenge Flowyager, we task it to execute these queries for a full day for all sites in the IXP-2019-09 dataset. We evaluate three different ways of answering the queries using Flowtree, namely using FlowQL with Flowtrees and 15-minute, 1-hour, and 1-day aggregation. On the IXP machine, we execute each benchmark 10 times and measure, just as before, the wall time as reported by the C++ chrono library.

Figure 3.12 shows the resulting FlowQL query response times for each benchmark as boxplots. Hereby, we distinguish between cold and hot query response times. In the hot case, relevant Flowtrees may be retrieved from the in-memory cache. In the cold case, we restart the in-memory cache process for each benchmark. If we use the 1-day Flowtrees, see Figure 3.12(a), the answers are readily available and the response arrives in the blink of an eye (less than 1 second). By using the in-memory cache we speed up query response time by about 10 to 50%. We also check the accuracy of the results and find that the results are accurate[3].

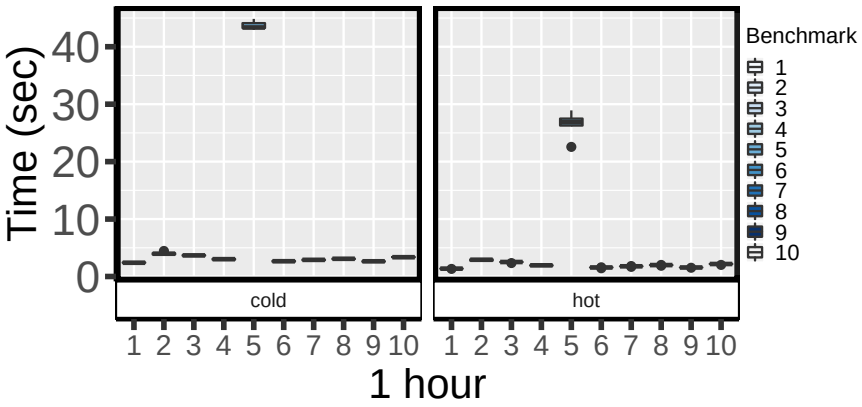
[3]We exclude Benchmarks 2, 5, and 9 as these benchmarks concern 60 min time-intervals and, thus, cannot be answered using data at 1-day granularity.

Table 3.5: Benchmark queries for Flowyager evaluation. Note that these queries correspond to those identified in Table 3.1.

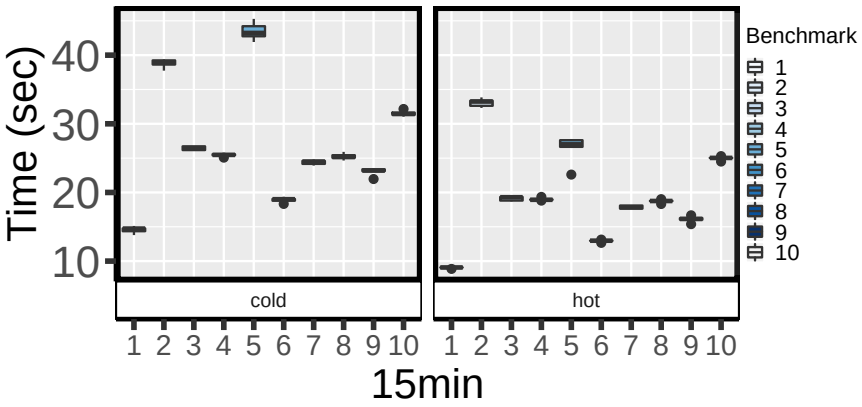
	Benchmark	Goal	Query
1	Aggregated flow statistics	Computing total traffic with specific features from IP/ports/time/location	SELECT pop(PROTO,COUNTMODE[,BIN]) FROM (time YYYY-MM-DD hh:mm to YYYY-MM-DD hh:mm) WHERE (site_id = ANY and dst_ip = IP/mask and dst_port = port/portmask)
2	Counting traffic	Computing Traffic volume between given IP/Port subnet/addresses, for a specific site n	SELECT pop(PROTO,COUNTMODE[,BIN]) FROM (time YYYY-MM-DD hh:mm to YYYY-MM-DD hh:mm) WHERE (site_id = n and src_ip = IP/mask)
3	Traffic flows	Displaying flows belonging to given subnets / IP addresses, passing through a specific site	SELECT *(PROTO,COUNTMODE[,BIN]) FROM (time YYYY-MM-DD hh:mm to YYYY-MM-DD hh:mm) WHERE (site_id = n and src_ip = IP/mask)
4	Traffic matrix	Finding popular flows from a subnet to subnets for all sites	SELECT above(K,PROTO,COUNTMODE[,BIN]) FROM (time YYYY-MM-DD hh:mm to YYYY-MM-DD hh:mm) WHERE (site_id = ANY and src_ip = ANY and dst_ip = ANY)
5	DDoS diagnosis	Finding the src IPs from which a dst IP (victim) has received abnormal traffic.	SELECT top(K,PROTO,COUNTMODE[,BIN]) FROM (time YYYY-MM-DD hh:mm to YYYY-MM-DD hh:mm) WHERE site_id = ANY and dst_ip = [victim_ip]
6	Super-spreader Detection	Finding hosts that send packets to more than k unique dst during a time interval (requires multiple queries)	SELECT above(K,PROTO,COUNTMODE[,BIN]) FROM (time YYYY-MM-DD hh:mm to YYYY-MM-DD hh:mm) where (site_id = ANY and src_ip = ANY) SELECT * FROM (time YYYY-MM-DD hh:mm to YYYY-MM-DD hh:mm) where (site_id = ANY and dst_ip = [pop_ip])
7	Top-k flows	Detect Top K flows in one or more sites , going to / coming from a specific subnet or IP address	SELECT top(K,PROTO,COUNTMODE[,BIN]) FROM (time YYYY-MM-DD hh:mm to YYYY-MM-DD hh:mm) WHERE site_id = n and (src_ip = IP/mask or dst_ip = IP/mask)
8	Heavy Hitters	Detect all flows with popularity over threshold T, in one or more sites, going to / coming from a specific subnet or IP address	SELECT hhh(T,PROTO,COUNTMODE[,BIN]) FROM (time YYYY-MM-DD hh:mm to YYYY-MM-DD hh:mm) WHERE (site_id = n and src_ip = IP/mask)
9	Heavy Changers Detection	Detect Top K heavily changed flows in one (or more) site(s).	SELECT hc(K,PROTO,COUNTMODE[,BIN]) FROM (time YYYY-MM-DD hh:mm to YYYY-MM-DD hh:mm)(time YYYY-MM-DD hh:mm to YYYY-MM-DD hh:mm) WHERE site_id = n
10	Full/4/5 tuple queries	Counting / Detecting flows belonging to a specific protocol/application	SELECT *(PROTO,COUNTMODE[,BIN]) FROM (time YYYY-MM-DD hh:mm to YYYY-MM-DD hh:mm) WHERE site_id = n



(a) 1-day Flowtrees (cold/hot cache)



(b) 1-hour Flowtrees (cold/hot cache)



(c) 15-minute Flowtrees (cold/hot cache)

Figure 3.12: IXP-2019-09: Flowyager response times (Table 3.5 benchmarks).

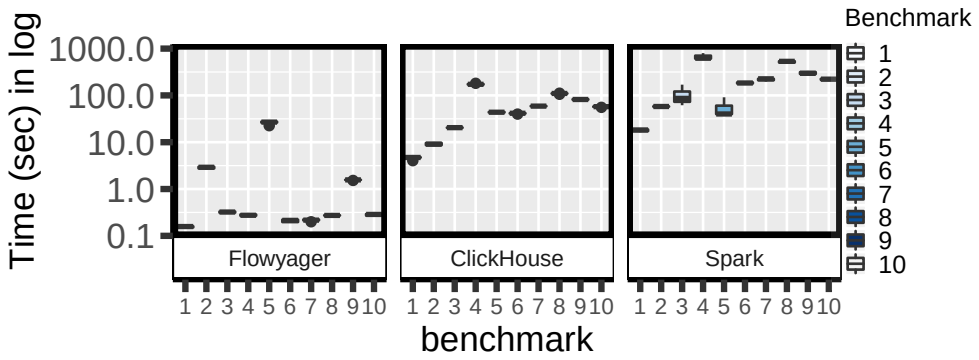


Figure 3.13: IXP-2019-09: Query response time comparison: Flowyager vs. ClickHouse vs. Spark (Table 3.5 benchmarks).

With 1-hour trees, see Figure 3.12(b), the query response times typically increase by roughly a factor of 7, even though the number of Flowtrees that have to be processed increases by a factor of 24. This is possible as Flowyager takes advantage of parallelization. For Benchmark 5 the query response time is the worst as we have to execute an iterator across all 24 hours. Note, this is no principle limitation of the design of Flowyager but a limitation of the implementation which does not yet parallelize the iterators. If we move to 15-minute trees, see Figure 3.12(c), the query response time increases further up to a factor of eight. This highlights the efficiency obtained by using higher granularity trees in the design of Flowyager. Note, all benchmarks are executed using a research prototype rather than a production system.

Using an appropriate Flowtree granularity, we can answer all except one benchmark query in less than 5 seconds, underlining that Flowyager is indeed able to answer apriori unknown queries. This query response time enables interactive exploration of the data.

3.6.3 Flowyager vs. Possible Alternatives

Finally, we explore how well Flowyager performs compared to other systems. We picked three alternatives, namely, using (a) task-specific data-parallel Python scripts, (b) Spark [38]–a state of the art data analytics platform, and (c) ClickHouse [81]–a state of the art column database. We evaluated all these systems on the same machine and dataset in IXP as previously described in 3.5.

First, we find that coding a custom python script for each benchmark takes a reasonably experienced programmer at least 2-3 hours for programming and debugging even if they can build upon a template from another benchmark. After all, it takes time to validate that the script is indeed doing what it is supposed to do. For some of the advanced tasks, e.g., the HHH, we did not start from scratch but rather included existing code. Nevertheless, this again did take additional time. Running the Python code on a day of data did take a mean of 39 minutes using a parallelization across 24 cores. Using 24 cores enables the script to parallelize the tasks by processing each

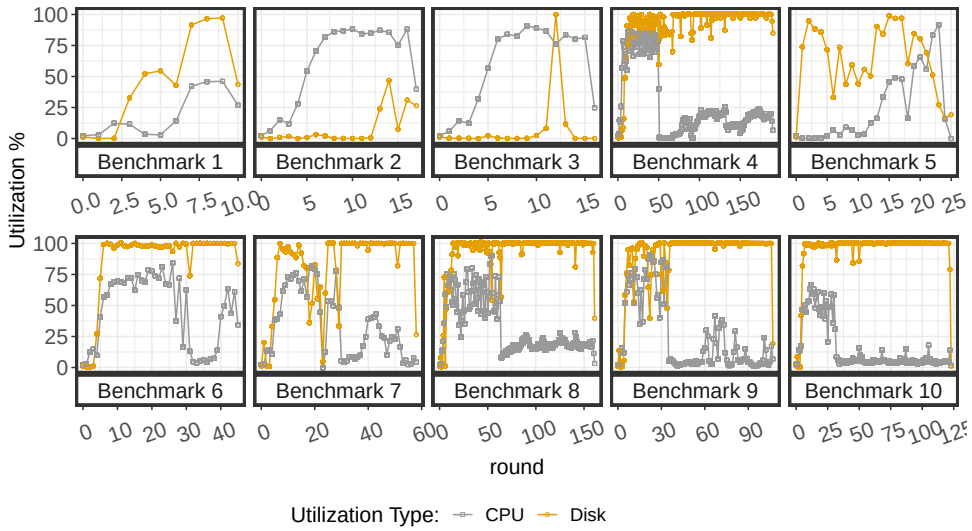


Figure 3.14: CPU and disk I/O usage in Spark experiments

hour of data in a separate process. Across all benchmarks, the Python code needed a minimum of 19 minutes and a maximum of 54 minutes.

Second, we find that setting up Spark and coding the queries require significant time. Indeed, it is necessary to first convert the data into a Spark-compatible format to get any reasonable performance (query response times less than 1 hour). This takes roughly 15.5 minutes per day of data for the IXP site. The resulting benchmark query response times are shown in Figure 3.13. Using this preprocessed data as input, the benchmark queries take a minimum of 20 seconds and up to 800 seconds. Note that for Benchmark 8 Spark only computes heavy hitters rather than HHH as implementing HHH on top of Spark is non-trivial. To measure the CPU usage and disk I/O usage of each Spark benchmark, we used the *iostat* command sampling every 5 seconds. In Figure 3.14, the x-axis shows the round, i.e., the 5-second period in which we sample, and y-axis shows the utilization in percentage. CPU utilization is shown in square points, while the round points show the disk I/O. We observe that in the majority of the cases, Spark is bound by disk I/O rather than CPU. This holds for benchmarks 1, 4-10. However, benchmark 2 is a drill-down query and requires multiple GROUPBY statements. Also, benchmark 3 works with only two features. Hence, the intermediate results are not too big to require frequent disk access. Therefore, unlike other benchmarks, benchmark 2 and 3 are limited more by CPU capacity than disk I/O. Indeed, this figure highlights the significant overhead of query processing using only the raw data.

Third, we set up an instance of ClickHouse. Here, it is necessary to first load the data into the database. This takes roughly 45 minutes per day of IXP-2019-09 data. On the other hand, the resulting benchmark query response times are significantly smaller than those of Spark, see Figure 3.13. Again, ClickHouse only supports a limited version of the HHH query for Benchmark 8. Figure 3.13 also includes the

Flowyager benchmark results from Section 3.6.2. Flowyager's benchmark performance supersedes all comparison systems.

3.6.4 Summary and Flowyager Limitations

Overall, Flowyager by far outperforms all three alternatives. Moreover, Flowyager is adaptive and supports HHH and physically distributed execution. We acknowledge that creating all Flowtrees does add some overhead–one day does take roughly 4 hours. However, this is a one-time operation, and overhead only matters if we consider archived data, but the Flowtrees can well be generated as the flow captures arrive. Moreover, it is easy to do memory management within Flowyager; e.g., rather than purging older data, we can summarize it.

The limitation of Flowyager is that its answers are only estimates. However, these are accurate both for elephants and mice flows alike. Hereby, we want to point out that most network-wide systems anyhow rely on highly sampled flow captures. As such the fact that we "only" provide estimates does not increase the uncertainties dramatically. If higher accuracy is necessary, we recommend combining Flowyager for data exploration with ClickHouse for focused in-depth analysis. Moreover, the insights from Flowyager can be used to instantiate online non-sampled queries using streaming network telemetry systems, such as Sonata [35].

3.7 Use-Cases

In this section, we showcase how to use Flowyager for tackling typical network operator tasks.

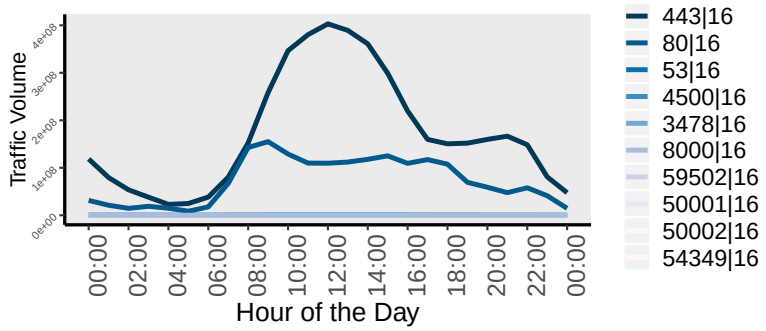
Unveiling Application Trends: With Flowyager we can easily infer the 10 most popular applications within a time period using a top10 query with site_id=ANY and src_port=ANY:

```
SELECT top(10,any,byte)
FROM (time 2018-05-09 00:00 to 2018-05-09 23:59)
WHERE site_id=ANY and src_port=ANY
```

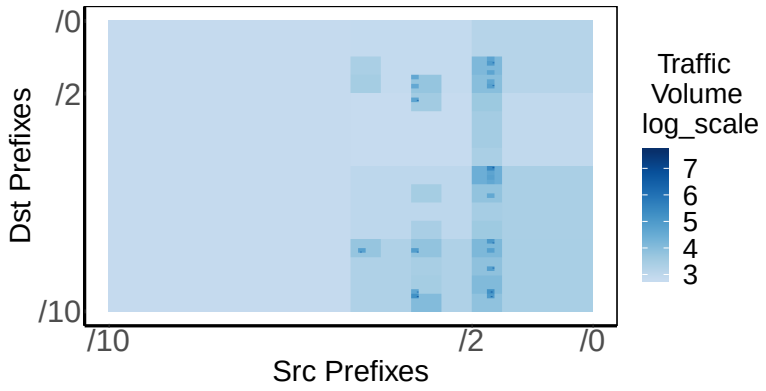
To then see how the popularity of each top 10 port changed over time we use the query pop-bin60 for each port. Therefore, the query would be:

```
SELECT pop(any,byte,bin60)
FROM (time 2018-05-09 00:00 to 2018-05-09 23:59)
WHERE site_id=ANY and src_port=X
```

See Figure 3.15(a) for the results for the MAWI-2018-05 dataset. We use the MAWI-2018-05 dataset for reproducibility as we will release the sample queries and their output along with the code. The query takes less than 1.4 seconds. Web and DNS related ports 80, 443, and 53 dominate. The same is true for the ISP. Still, during peak, other port numbers are prominent as well, e.g., port 3074. This port is used



(a) Application trends



(b) Traffic matrix.

Figure 3.15: MAWI-2018-05: Data exploration

by Xbox LIVE and Games for Windows–Live. The peak traffic time also is the peak activity time for gaming, at least for residential customers of this Tier-1 ISP.

Traffic matrix: Computing a traffic matrix involves determining all src/dst pairs with a traffic volume larger than a value X. With Flowyager, one can use the above_t, for src_i=ANYp and dst_ip=ANY. Therefore, the following query can be used:

```
SELECT above(X,udp,byte)
FROM (time 2018-05-09 00:00 to 2018-05-09 23:59)
WHERE site_id=ANY and src_ip=ANY and dst_ip=ANY
```

To highlight this capability we determine the src/dst traffic matrix for the MAWI-2018-05 data, see Figure 3.15(b). It shows the traffic matrix at different aggregation levels to detect which pairs of the source (src) and destination (dst) prefixes (at different granularity levels) are responsible for a large fraction of traffic exchange. For visualization, we use a two-dimensional heatmap where the x-axis corresponds to src IPs, the y-axis to dst IPs, and the color to the traffic volume normalized by the number of IPs within the area, i.e., traffic flowing from a src prefix to a dst prefix. This query took less than 13 seconds.

Investigating DDoS attacks: Network attacks, and in particular, distributed denial-of-service (DDoS) attacks are an ongoing nuisance for network operators as well as network users. A large body of research papers has focused on techniques

for detecting DDoS attacks, see, e.g., [108–111], including references and citations. Indeed, the multitude and the impact of DDoS attacks, see, e.g., [112, 113], have given rise to a variety of different mitigation techniques, see e.g., [114, 115]. Still, detecting DDoS attacks reliably as well as diagnosing their root causes is critical for starting countermeasures or taking preventive future actions. Flowyager is an ideal system for tackling this challenge.

One of the most common signatures of DDoS attacks is a sudden rise in traffic for src/dst ports that are used within amplification attacks [112, 116–118]. Among such ports are 0, 123 (NTP), 11211 (memcached), 53 (DNS), and 1900 (SSDP), as discussed above. Potential DDoS attacks can be found by using the heavy changer query. It identifies time ranges during which they occurred. We execute these queries for each hour:

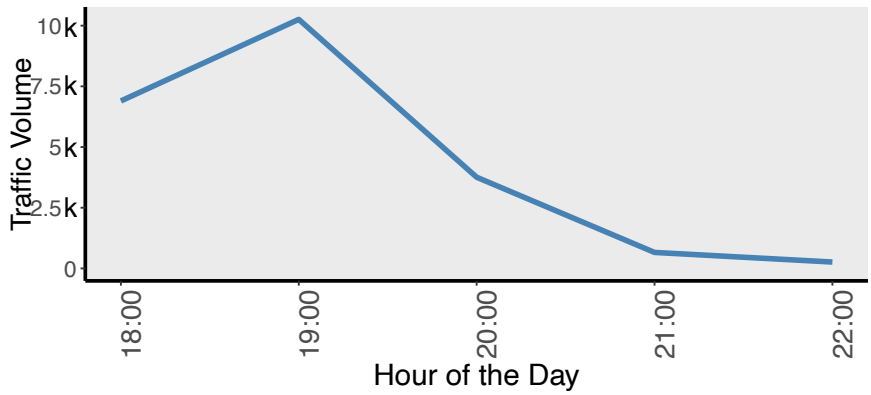
```
SELECT hc(100,any,byte)
FROM
  (time 2019-04-01 00:00 to 2019-04-01 00:59)
  (time 2019-04-01 01:00 to 2019-04-01 01:59)
WHERE site_id=ITR and (dst_port=ANY or src_port=ANY)
```

Per hour this takes less than 0.3 seconds. Among the heavy changers are high volume ports related to Web traffic, i.e., port 80, 443, as well as other ports where the volume can easily vary. But, we also find some unusual ports, i.e., 123 (NTP) which are known to be involved in DDoS attacks. Figure 3.16 shows a DDoS amplification attack in one of the sites of the ISP. This is a DDoS attack on NTP (port 123). Here, a very large number of src IPs scattered across multiple networks are involved but only a few dsts are targeted; namely two, whereby one of them receives more than 95% of the attack packets. It took us less than 5 minutes of human time and less than 1 minute computation time to find the attack for port 123, the site, the src of the attacks, and identify the start and the end of the attack. To illustrate the exploratory power of Flowyager, we identified the hours where the attack took place, see Figure 3.16(a), within a second. Then, we drill-down to the 15 minutes granularity to infer the start and end of the attack, see Figure 3.16(b), with a second query that took two seconds of execution time:

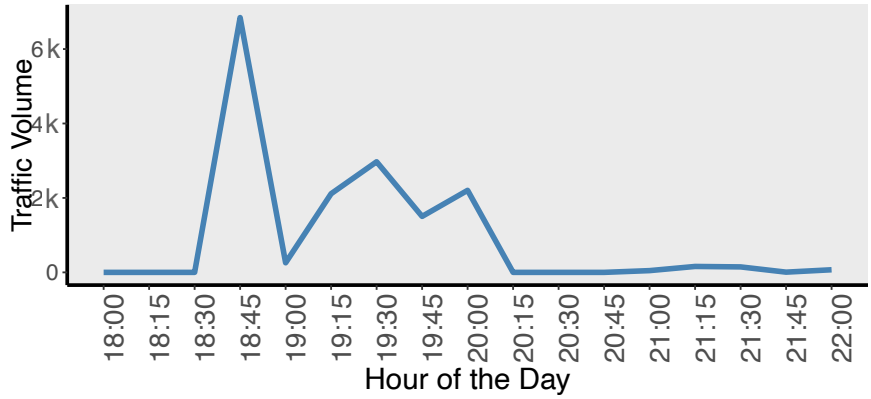
```
SELECT pop(any,byte,bin15)
FROM (time 2019-04-01 01:00 to 2019-04-01 01:59)
WHERE site_id=ITR and dst_port=123|16
```

Note, detecting slowly increasing DDoS attacks needs a different approach. Here, a diff query to an earlier time period can be used as an indicator.

Towards real-time DDoS Mitigation: Using insights from historical analysis of DDoS attacks it is possible to use Flowyager also for near-live analysis if we keep recent Flowtrees at a shorter time granularity, e.g., 1-minute bins: we can then either use the above queries to monitor ports highly affected by DDoS attacks or we can use heavy-changer queries to look for ports with unusual activity. If we see such unusual activity, we can use the drill-down capabilities of Flowyager to check if, e.g., the traffic is targeted at specific IPs, i.e., only involves a small number of src or dst addresses, or involves spoofed addresses, i.e., a large number of IP addresses. If yes, Flowyager can



(a) ISP-2019-04: 123 port activity across 60 minute time bins



(b) ISP-2019-04: 123 port activity across 15 minute time bins

Figure 3.16: ISP-2019-04: DDoS NTP attack investigation

be used to trigger an alarm which may then blackhole the attack traffic, e.g., using a system such as Stellar [114] or traffic scrubbing systems [113]. Recall that other techniques, e.g., telemetry, need to know a-priori the queries they have to execute. The power of Flowyager is that is can answer arbitrary queries that are not known in advance and using the already available network flow summaries supported by router vendors. Thus, Flowyager offers security capabilities that can help to identify arbitrary security issues. It can also help in generating the appropriate queries to execute them in real-time when, e.g., telemetry is used.

Lessons Learned: For our use cases neither the initial sampling in the flow captures nor the Flowyager estimates were detrimental to achieving the goal. However, we noticed some implementation challenges, e.g., handling flows from routers with unsynchronized clocks. We decided to use the timestamp when the flow is arriving at FlowAGG. Note that this may lead to some small amount of misbinning if the router is distant (in terms of network delay) from the aggregator. However, the impact is expected to be limited and probably well within the typical uncertainty of flow captures. Note that our approach even enables us to update Flowtrees of past time bins, should a significant number of flows arrive delayed.

Another observation is that one can tune Flowyager according to the needs of the users. Overall, we find that a query can be answered quickly if the aggregation level of the available (cached) Flowtrees matches the query granularity in terms of site sets and/or time granularity. The reason is that this avoids merging Flowtrees on the fly. Thus, if many queries involve the same subset of interfaces, e.g., per router, or all long-haul interfaces, it may make sense to store additional Flowtrees, if only temporarily. For example, keeping a Flowtree for all sites adds little overhead but speeds up queries significantly.

3.8 Summary

In this chapter, we designed, developed, and evaluated Flowyager, a system that allows exploration of network-wide data and answering ad-hoc a priori unknown queries within seconds. It achieves this using already existing network flow captures, without the need for specialized hardware, and without the need to compile specific queries into telemetry programs that should be known in advance and are slow to update.

Flowyager uses succinct summaries, Flowtrees, of raw flow captures and provides an SQL-like interface, FlowQL, that is easily usable by network engineers. We showcase the performance and accuracy of Flowyager in two operational settings: a large IXP and a tier-1 ISP. Our results show that the query response time can be reduced by an order of magnitude, and, thus, Flowyager enables interactive network-wide queries and offers unprecedented drill-down capabilities to identify the culprits, pinpoint the involved sites, and determine the beginning and end of a network attack.

4

FlowDNS: Correlating Netflow and DNS Streams at Scale

In this chapter, we focus on identifying domain names associated with Internet traffic flows to contribute to the overarching goal of uncovering hidden characteristics of Internet traffic. ISPs need to know where their traffic is coming from originally not only to provide their customers with better quality but also to better plan their network infrastructure and collaborations. As explained in Chapter 2, most of the service providers gather network-layer statistics of their traffic using different protocols, e.g., Netflow [119] and IPFIX [120] which usually include source and destination addresses, and traffic volume. Network-layer headers do not contain the domain name of the services they belong to. To complicate this even more, Over-The-Top (OTT) services are nowadays adopting multi-CDN approaches [121], making the inference of the service merely on the IP address nearly impossible. Therefore, either DNS records or the application layer information are needed. Nowadays, the application layer information is oftentimes encrypted [122] and therefore, not visible to the service providers.

DNS is one of the core services to map domain name to the IP address [123, 124], and can be used to find the original source of the service. There have been several studies using machine learning techniques for domain name recognition, all of which use passive DNS records [125–129]. Inspecting the traffic on a large European ISP, we observe that more than 85% of the traffic is originated by CDNs. These CDNs might use one domain name for several services in different locations/times and IP addresses could also be re-used [130]. Therefore, IP address to domain name mappings change frequently in CDN-hosted domain names [130, 131], and DNS records used for such correlation should not be outdated. Thus, capturing the DNS records collected from user requests is the most suitable way of mapping domain name to IP address.

Previous studies use software-defined networking to correlate DNS responses with web traffic either in an external controller [132] or in the data plane [133, 134]. However, all these approaches introduce parsing limitations, e.g., domain names length limit, and also ignore encrypted DNS packets. This work, however, does not enforce any limitation on the DNS records, and is not affected by DNS encryption. Unlike the previous studies, we do not aim at direct policy enforcement and therefore, do not require any modification to the existing network architecture. We instead propose a system that can run on any machine receiving a flow of DNS records and network flows.

Using DNS records from the same source as the traffic gives the domain name recognition more certainty. In the meanwhile, using the same sources of DNS and Netflow translates to a higher processing load since both sources need to be processed synchronously either in a real-time fashion or offline. In case the processing is to be done offline, the timestamps need to be taken into account and the two sources of data, namely Netflow and DNS records, need to be correlated in the window where the DNS record is still valid, i.e., TTL > 0. Although research has shown that in some scenarios, longer TTLs can reduce latency [135], our experiments show that 99% of the record have TTLs of less than two hours. Monitoring TTLs for every single record while correlating induces higher memory usage and lower correlation rates. Multi-level caching in DNS resolvers makes this even more complicated [136].

Correlating two live sources of data, each carrying thousands or millions of records per second, also requiring keeping some of the records for later use, is not trivial. Doing so with the standard database queries for an hour of data, takes tens of hours, making this correlation impossible to be done in real-time. Therefore, we propose FlowDNS, a system to correlate Network-layer headers and DNS records in real-time. Our work consists of three main contributions:

- We design, build, and deploy a system for real-time DNS-Netflow Correlation called FlowDNS in a large European ISP. In Section 4.2, we go over the system's building blocks, and in Section 4.3, we show that we can correlate 81.7% of the data.
- Using the correlated data in our deployment of FlowDNS, we identify the traffic using malformatted, spam and phishing domains in Section 4.4. We observe that 0.5% of the daily traffic volume uses either malformatted or spam/phish domain names.
- Finally, we formulate our learned lessons in building a real-time DNS-Netflow Correlation system in Section 4.5.

4.1 Data Overview

We use flow data and DNS traffic from a large European ISP. The flow data is in Netflow format, and we receive both Netflow and DNS traffic as live streams:

- **DNS streams**: A set of DNS *cache misses* gathered from different customer resolvers. For load-balancing purposes, the data is already divided into 2 different streams, carrying 75K DNS records per second on average collectively. Each record in a DNS stream contains:
  timestamp,..., [name; rtype; ttl; answer] <0,n>
- **Netflow streams**: A set of Netflow records captured at the network ingress interfaces. For load-balancing purposes, the data is already divided into 26 different streams. These streams input 1M Netflow records per second on average. As also explained in Section 2.3.1, each record in a Netflow stream contains:
  ..., srcIP, dstIP, ..., timestamp, packet, bytes

We also deploy our system on a smaller European ISP with one DNS stream carrying 115K DNS records per second and two Netflow streams with 138K Netflow records per second.

Each of the above-mentioned streams has an internal buffer to be used in case the reading speed is less than their actual rate. If that buffer overflows, the streams start to drop data. Throughout this paper, wherever we mention *loss on the streams*, we mean that the buffers are overflown and start to drop. Therefore, the goal is to keep the buffer usage stable to avoid any loss.

Reading from multiple streams requiring shared memory access, and keeping the DNS records in memory to be quickly accessible makes this correlation challenging in terms of memory and CPU usage. To overcome these challenges, we leverage different techniques, details of which are explained in Section 4.2. In accordance with the data provider agreement, we refrain from reporting the exact values of the traffic, and all the traffic volume data throughout the paper is normalized.

4.2 Methodology

The goal is to categorize source of the traffic by their service in near real-time, e.g., to understand what fraction of the traffic is originated from Netflix, Amazon Prime, Google, etc. To realize this, we look for the IP address of the Netflow records in the *answer* section of the A/AAAA DNS records to find the *name* it corresponds to. Then looking at the CNAME records, we search for that *name* to find the corresponding CNAME. The results from this correlation are then correlated with BGP information to find the information such as source/destination ASes for each service. In this chapter, we only focus on DNS-Netflow correlation since BGP correlation is out of the scope of this work. We note that the system is not bound to NetFlow data and can be adapted to use other data formats containing IP addresses and timestamps in a configuration file.

4.2.1 Overview

To perform the DNS-Netflow correlation, as Figure 4.1 shows, the DNS streams are received by *FillUp* workers. Multiple FillUp workers are allocated to each DNS stream to enable parallel processing of each shard of the DNS stream. These workers analyze the DNS records and fill up a shared internal storage with the DNS records. At the same time, the Netflow streams are received by the *LookUp* workers. These workers look for the source of the traffic in the shared internal storage. Again, we assign multiple LookUp workers to each Netflow stream. In our work, we are interested in analyzing the source of the traffic, hence we use the source IP address. Nonetheless, destination address or both source and destination addresses can be used with minor modifications. Then the result of this lookup is written onto the disk by the *Write* workers. Each worker has an input and output queue which enables the communication between workers. It is important to avoid that too many workers

write to the same queue, as this contention causes a decrease in performance. Since multiple instances of the LookUp workers will try to simultaneously access a shared data structure where DNS records are kept, we split the DNS data and distribute them to different splits to then isolate each split as much as possible. In Section 4.3, we discuss whether this further splitting is needed. In Section 4.2.2 and Section 4.2.3, we go through the steps starting from reading the streams, to fill up the internal storage, and then look up and write.

We cannot strictly apply DNS records TTL and expire them after the TTL has passed, since there are multiple levels of caching and each might apply a different tolerance for expiring the records. Moreover, applying the actual TTLs on the DNS records requires regular iterations over all DNS data to check their TTL expiration. This degrades the performance dramatically and increases the memory usage. We tried applying the exact TTLs from the DNS records on our correlation, meaning we correlate the IP from a DNS record with the source IP from the Netflow record only if the DNS record's TTL plus its timestamp is less than the timestamp from the Netflow record which we consider current timestamp. In other words:

$$TTL_{\text{dns}} + Timestamp_{\text{dns}} < Timestamp_{\text{netflow}}$$

We also run a regular process to clear-up the expired DNS records, when the above-mentioned condition did not hold. We run this on the same sources of data, meaning DNS and Netflow streams at the large European ISP, and observed that the internal buffers of all the streams start to overload from the very first minutes of running the above-mentioned system, with the loss rate of over 90% for both Netflow and DNS streams. We observed that the memory usage reaches up to 45 GB memory usage after only 1 hour of running the system. Comparing this to the results from FlowDNS in Figure 4.4(b) which we will explain thoroughly in Section 4.3, we observe that the memory usage is doubled although only 10% of the data is received at the system and others are lost. This could be due to the regular clear-up process not being fast enough to clear-up all the expired TTLs as the hashmaps grow, while at the same time, the contention to access the shared memory is so high that the performance degrades dramatically.

On the other hand, we cannot keep the DNS records forever due to memory constraints. Therefore, we need to clear up the storage. We observe that 99% of the A/AAAA and CNAME records have a TTL smaller than 3600 and 7200 seconds, respectively (Ref. Figure 4.2). Therefore, we clear up the A/AAAA records every 3600 seconds, and CNAME records every 7200 seconds. However, since clearing the whole storage will remove all the states, we perform buffer rotation before clear-up. The internal storage where we keep the DNS records is a hashmap with the DNS *answer* section as key, and the *query name* as value. For implementing these hashmaps, we use the *concurrent-map* module in Go [137], which allows for high-performance concurrent reads and writes by sharding the map.

We add the DNS records in a primary hashmap, i.e., the *active* hashmap. After a certain amount of time has passed, we copy the contents of the active hashmaps into a secondary storage, i.e., the *inactive* hashmap, and clear up the active hashmap. In the

Parameter Name	**Description**
AClearUpInterval	Time in seconds after which the *IP-NAME* hashmap is cleared.
CClearUpInterval	Time in seconds after which the *NAME-CNAME* hashmap is cleared.
NUM_SPLIT	Number of splits for each IP-NAME hashmap.
Storage Name	**Description**
IP-NAME$_{\text{Active n}}$	Hashmap for DNS records with TTL < AClearUpInterval and label n.
IP-NAME$_{\text{Inactive n}}$	Hashmap where the contents of IP-NAME$_{\text{Active n}}$ are copied to every AClearUpInterval seconds
IP-NAME$_{\text{Long n}}$	Hashmap for the new DNS records with TTL >= AClearUpInterval and label n.
NAME-CNAME$_{\text{Active}}$	Hashmap for the new CNAME responses with TTL < CClearUpInterval.
NAME-CNAME$_{\text{Inactive}}$	Hashmap where the contents of NAME-CNAME$_{\text{Active}}$ are copied to every CClearUpInterval seconds
NAME-CNAME$_{\text{Long}}$	Hashmap for the new CNAME responses with TTL >= CClearUpInterval.

Table 4.1: Overview of FlowDNS parameters and storage names

next clear-up round, the current contents of the inactive hashmap will be over-written by the new contents. Active hashmaps are actively updated with the newly arrived DNS records, while inactive hashmaps are only updated when the active hashmaps are cleared. A very small fraction of the DNS records has TTLs longer than the clear-up interval. Therefore, in case a DNS record's TTL is larger than a certain threshold, we put it into specific hashmaps which are never cleared or are cleared much less frequently, namely, long hashmaps. Otherwise, it is stored in the active hashmap. From now on, we call the active/inactive hashmaps for A/AAAA and CNAME records IP-NAME$_{\text{active/inactive}}$, and NAME-CNAME$_{\text{active/inactive}}$ respectively. Table 4.1 shows an overview of the parameters and names of the in-memory storage that we use in FlowDNS. Note that $0 \leqq n < NUM\_SPLIT$.

4.2.2 DNS Processing

This part of FlowDNS takes in DNS streams and fills up an internal shared storage with the DNS records. These records will then be accessed in the Netflow Processing.

1. DNS Streams are received by separate threads.
2. The DNS records go through a filter to check if they are valid DNS responses.

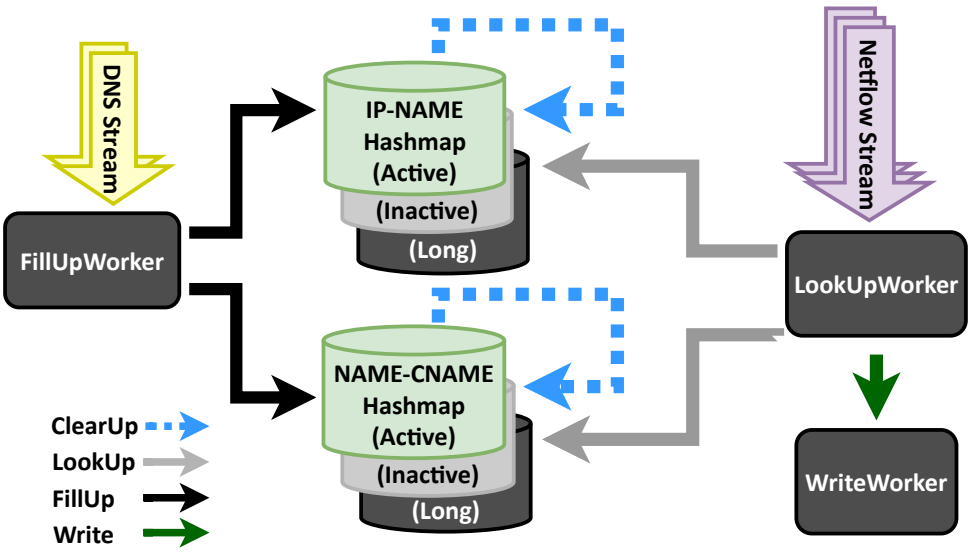


Figure 4.1: FlowDNS correlation architecture

3. Valid DNS responses are added to a queue, namely FillUp Queue, to be then processed each by several FillUp workers. We need this queue to facilitate the synchronous execution of different workers.
4. Each FillUp worker picks a DNS response from the FillUp Queue and if it is an A/AAAA record, labels it based on the IP address. This label will be used as a hashmap index later on.
5. The FillUp worker then puts the DNS response in the shared hashmaps. In all our hashmaps, the key is the answer section, and the value is the query. We leverage two kinds of hashmaps:

- **IP-NAME hashmap:** Maps the answer section in an A/AAAA response, i.e., the IP address, to the queried domain name. We divide these hashmaps into several splits. We empirically find that 10 splits are suitable for our scenario. This can change for any deployment depending on the traffic volume. If, in Step 4, the IP for an A/AAAA response gets the label n, $0 \leq n < 10$, it goes to $IP\text{-}NAME_n$.
- **NAME-CNAME hashmap:** Maps the answer section, i.e., the domain name, to the queried canonical name for CNAME records.

6. The FillUp worker keeps track of the timestamp in each DNS record. If *AClearUpInterval* seconds has passed, it copies the contents of $IP\text{-}NAME_{active}$ to $IP\text{-}NAME_{inactive}$ and clears the $IP\text{-}NAME_{active}$. If *CClearUpInterval* seconds is passed, it copies the contents of $NAME\text{-}CNAME_{active}$ to $NAME\text{-}CNAME_{inactive}$ and clears the $NAME\text{-}CNAME_{active}$.

Algorithm 4 shows an overview of the fillUpWorker thread which which the DNS records and fills in the hashmap.

To find out the correct number for the clear-up intervals, namely *CClearUpInterval* and *AClearUpInterval*, we investigate the TTLs for the DNS records. We look at

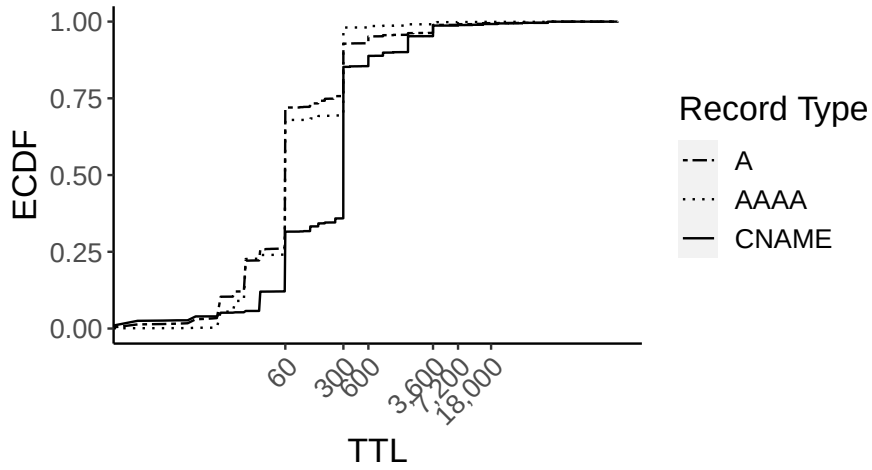


Figure 4.2: Cumulative distribution of TTLs for DNS records over a day

the DNS records' TTLs over a day at a large Europoean ISP, and find out that 99% of the A/AAAA and CNAME records have TTL smaller that 3600 and 7200 seconds respectively, shown in Figure 4.2. Therefore, in FlowDNS, we set the clear-up variables as follows:

$$CClearUpInterval = 7200$$

$$AClearUpInterval = 3600$$

4.2.3 Netflow Processing

In parallel with the DNS processing, this part of FlowDNS takes in the Netflow streams, takes the source IP address, and looks for this IP address in the internal shared storage to find the corresponding domain name. It then writes the results into the output files.

1. Netflow Streams are received by separate threads.
2. The Netflow records go through a filter to check if they are valid Netflow records.
3. The valid Netflow records are added to the LookUp Queue to be processed each by several LookUp workers.
4. Each LookUp worker picks a Netflow record from the LookUp Queue and labels it based on the *srcIP* field.
5. The LookUp worker looks for the *srcIP* in the $IP\text{-}NAME_{active\ n}$ hashmap, if the label from Step Item 4 is n. If nothing is found, it looks into the Inactive hashmap, and next into the long hashmap. If a *Name* is found, the search continues onto the next step. Otherwise, the search finishes here for that *srcIP* (*result* = NULL).
6. The search will be continued in $NAME\text{-}CNAME_{active\ n}$ to find the CNAME for that *Name*. If a *CName* is found, the search continues to the next step. Otherwise, the search continues in the Inactive and then the long hashmap. If nothing is found, the search finishes here for that *Name* (*result* = *Name*).

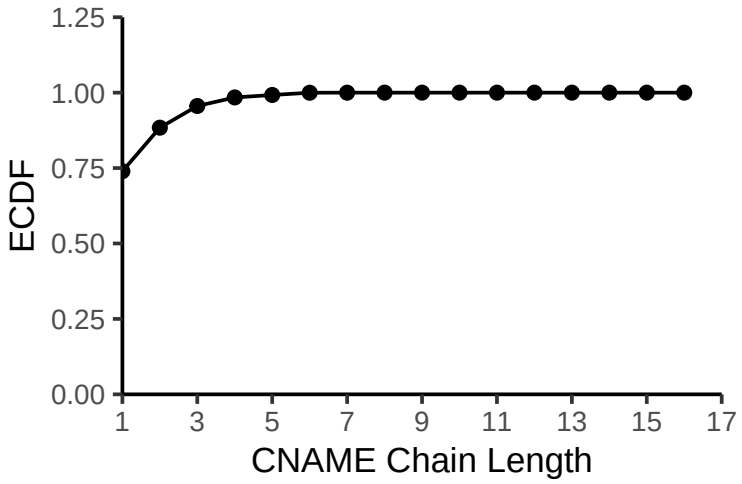


Figure 4.3: Cumulative distribution of CNAME chain length over a day

7. The search in the NAME-CNAME map continues until no further CNAME is found or a pre-defined loop limit is reached (*result = CName*). CNAME look-up can sometimes include multiple consequent look-ups with one CNAME mapping to another more than once. We studied the CNAME chain length, and as shown in Figure 4.3, we observed that more than 99% of the DNS records can be mapped with a chain of 6 look-ups. Therefore, we limit the number of CNAME chain look-ups to 6 in FlowDNS.
If the *result* is found with more than one look-up in NAME-CNAME maps, we add it to $NAME\text{-}CNAME_{active}$ for later use.
8. The *result* along with the original Netflow is then passed to the Write Queue to be written in the output file by WriteWorkers.

Algorithm 5 shows an overview of the lookUpWorker thread which reads the netflow records, looks them up in the Active, Inactive, and Long hashmaps and finds the results.

We publish the code for FlowDNS [2] for future researchers or network operators.

4.3 Evaluation

In this section, we evaluate the matching accuracy and performance metrics for FlowDNS implemented in Go. First, we analyze the final version of FlowDNS on a full week of traffic of a large European ISP. Second, we selectively remove implementation features from FlowDNS on a one-day traffic capture to understand their importance by evaluating the effect on matching accuracy, CPU usage, and memory consumption. We evaluate all the benchmarks on an Ubuntu 18.04.5 LTS machine with 128 cores and 756 GB RAM. Figure 4.4 shows the CPU and memory usage of FlowDNS over one week when deployed at a large European ISP. In both plots,

Algorithm 4 DNS Read and Fill-up Overview

```
Function fillUpWorker DNSRecord d
    n = label(d)  if d.rtype is A/AAAA then
        if d.ts - lastAClearUpTs >= 3600 then
            IpName.Inactive = IpName.Active
            IpName.Active = {}
            lastAClearUpTs = d.ts
        end
        if d.ttl <= 3600 then
            IpName.Active[n][d.answer] = d.query
        else
            IpName.Long[n][d.answer] = d.query
        end
    else
        if d.ts - lastCClearUpTs >= 7200 then
            NameCname.Inactive = NameCname.Active
            NameCname.Active = {}
            lastCClearUpTs = d.ts
        end
        if d.ttl <= 7200 then
            NameCname.Active[n][d.answer] = d.query
        else
            NameCname.Long[n][d.answer] = d.query
        end
    end
end
```

the right axis shows the traffic volume to compare the CPU/memory usage patterns with the traffic load. For all three metrics–traffic volume, memory usage, and CPU usage–we can clearly identify diurnal patterns, with daily peaks in the evening period, a low time during night hours, and an increase during the day. Note that we normalize the traffic volume in the right Y-axis. We show CPU usage as percentages, i.e., every 100% means 1 fully utilized CPU core. The CPU usage is around 2500% which means roughly 25 CPUs are used. Memory usage also oscillates between 15 GB and 30 GB. In addition to the large European ISP, we also deploy FlowDNS on a smaller network. On the smaller network, we observe average memory usage of 6 GB, and CPU usage of around 300%, both following a diurnal pattern. The ratio of CPU usage and number of flows remains the same in both deployments. The memory usage, however, is affected by both number of DNS records and number of parallel threads. This results in lower memory usage in the smaller network. On both deployments, the results are written to disk by a maximum delay of 45 seconds, and without any significant loss, i.e., 0.01% loss, on the data stream buffers. The ratio of correlated traffic to the total traffic, i.e., the correlation rate, is 81.7% on average for both deployments. We cannot correlate 18.3% of the traffic since (1) the coverage of our DNS data is only 95%, as discussed later in this section, and (2) not all the traffic is DNS-related, i.e., not all traffic has the destination IP address obtained through a DNS query.

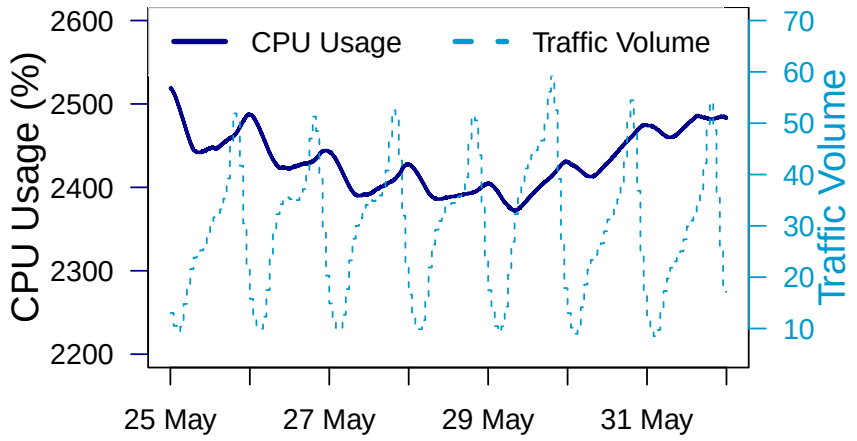
Algorithm 5 Netflow Read and Look-up Overview

```
Function lookUpWorker NetflowRecord nf
    n = label(nf)
    Name = deepLookUp(nf.srcIP, IpNameObj[n])
    loopCount = 0
    if Name != Null then
        results = append(results, Name)
        Cname = deepLookUp(Name, NameCnameObj[n])
        while Cname != Null and loopCount <= 6 do
            results = append(results, Cname)
            loopCount ++
        end
    return results
Def deepLookUp NetflowRecord nf, MapObj hm
    Name = NULL  if  nf.srcIP in hm.Active then
        Name = hm.Active[nf.srcIP]
    else if  nf.srcIP in hm.Inactive then
        Name = hm.Inactive[nf.srcIP]
    else if  nf.srcIP in hm.Long then
        Name = hm.Long[nf.srcIP]
    return Name
```

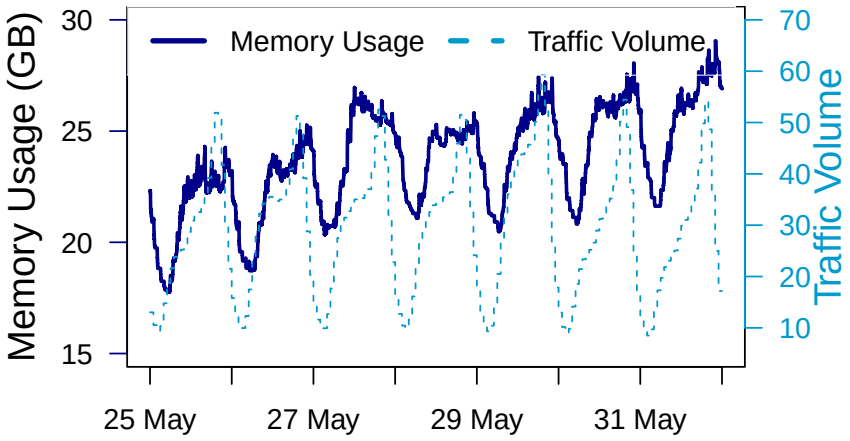
Now, we remove the techniques used in the fully featured version of FlowDNS once at a time, introducing four new benchmarks:

- *No Split*: The hashmaps are not divided into several splits.
- *No Clear-Up*: The hashmaps are kept in memory forever.
- *No Rotation*: The hashmaps are cleared, but no buffer rotation takes place and no Inactive hashmap exists.
- *No Long Hashmaps*: The hashmaps are cleared up, and buffer rotation takes place, but records with large TTLs are also written in the Active hashmaps, instead of the Long hashmaps.

Figure 4.5 shows CPU and Memory usage for the above benchmarks. As expected, memory usage for *No Clear-Up* grows steadily over the day and can easily hit the memory limit. The mean correlation rate for this benchmark is 82.8%. The *No Rotation* benchmark uses much less memory compared to other benchmarks since it does not keep a copy of the original contents before clear-up. However, the average correlation rate for this benchmark is 79.5%. The *No Long Hashmaps* save neither a significant amount of memory nor CPU, yet, with a correlation rate of 81.1%, reduce the correlation rate by 0.6% compared to the Main benchmark. Therefore, the Long Hashmaps help keep those DNS records from being cleared up without much cost. The *No Split* neither improves nor degrades the memory usage but decreases the CPU usage significantly. This could be due to the reduced effort to access separate hashmaps simultaneously. The average correlation rate is also 81.7%. However, this should not be interpreted as if sharding the data is not helpful at all. In contrast, as we have used data sharding already both in our hashmaps, and in our job queues,



(a) CPU usage



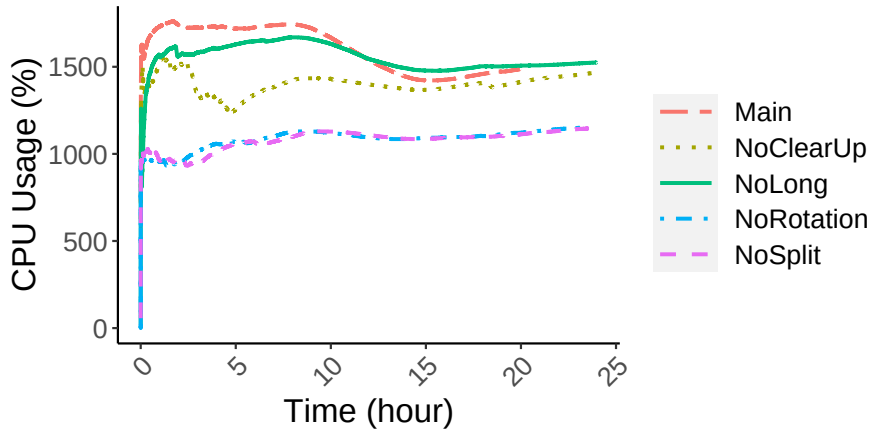
(b) Memory usage

Figure 4.4: CPU and memory usage for *Main* benchmark over a week

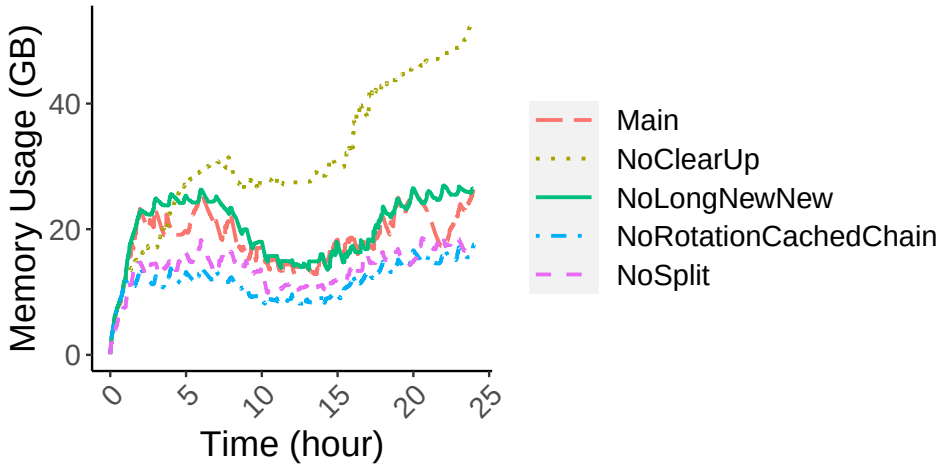
explained in Section 4.2.1. The fact that this feature does not help as much as the others only shows that no further splitting is needed in our case. The correlation rate, i.e., the ratio between correlated traffic and total traffic is illustrated in Figure 4.6 for different benchmark variants. The *No Split* benchmark excluded from the plot since it has a complete overlap with the *Main* benchmark. The top two variants in terms of correlation rate are *Main* and *NoClearUp*. The *NoClearUp* performs unacceptable in terms of memory usage. The lowest correlation rate belongs to *NoRotation*, which shows the importance of buffer rotation in FlowDNS.

As we have seen with these four benchmarks, all implemented features in FlowDNS, except for IP-splitting, help increase the correlation rate while keeping the CPU and memory usage low.

Coverage. FlowDNS receives DNS cache misses gathered from the clients' default ISP resolvers. This data is sent from the ISP resolvers to our collectors via TCP. Therefore, even if the client uses DNS encryption while still using the default ISP resolver, the results from FlowDNS are not affected. However, if the clients use



(a) CPU usage



(b) Memory usage

Figure 4.5: CPU and memory usage for different variants over a day

a resolver other than the default ISP resolver, e.g., a public DNS resolver (e.g., Cloudflare's 1.1.1.1, Google Public DNS, or Quad9), the DNS record is not received and therefore, FlowDNS can not correlate the Netflow traffic for those clients. To understand the number of DNS records we lose due to clients using public DNS resolvers, we analyze a sample 1-hour Netflow data and filter DNS and DoT traffic, i.e., ports 53 and 853. Then, using a public DNS resolvers list [138] and comparing it with our sample, we observe that 1 out of every 20 DNS packets is sent to a public DNS resolver. Therefore, the coverage of our DNS data is 95%.

Accuracy. Earlier in Section 4.3, we report that the correlation rate for FlowDNS, i.e., the number of bytes that could be correlated with *any* service/domain name compared to the total traffic volume in bytes, is 81.7% on average. However, this metric does not show whether the correlated service is the service actually used by the clients. Since we cannot access the actual domain names used by the clients, there is no ground truth against which we can compare our results. Nevertheless,

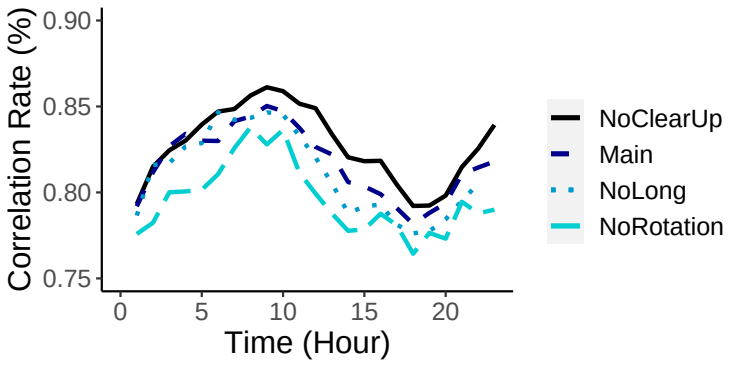


Figure 4.6: Correlation rate for benchmark variants

we can estimate the accuracy of FlowDNS, by pinpointing the scenarios that could result in an incorrect service correlation and estimate their impact on the system's accuracy.

In FlowDNS, we keep the DNS data in a hashmap with the IP address as the key and domain names as values. Therefore, by design, observing multiple IP addresses for one domain name in the DNS data does not affect the accuracy of FlowDNS. However, observing multiple domain names for one IP address can affect the accuracy. In case a second domain name is observed with the same IP, i.e., the same key, the existing (first) domain name is overwritten by the second domain name, which in turn decreases the accuracy of our system. To confirm this, we design a small-scale accuracy analysis using generated traffic data. We browse two different websites and capture the traffic. Then, we extract the DNS packets from the captured traffic and feed them to FlowDNS as the DNS stream. We then create Netflow records from all traffic packets and feed them to FlowDNS as the Netflow stream. Finally, observing the correlated domain names and comparing them to the actual scenario, we find whether the system has correlated correctly. We consider two scenarios for this experiment: (1) Two websites with different domain names and different IP addresses. (2) Two websites with different domain names, using the same IP address. In the first scenario, we observe that all the traffic is correlated correctly, while in the second scenario, all the traffic is correlated to the second domain name. In other words, we had an accuracy of 100% and 50% in the first and second scenarios, respectively.

To estimate the impact of such mislabelling events, we analyze the domain name distribution per IP address. To this end, we analyze a 300-second sample of DNS records since as Figure 4.2 shows, more than 70% of the DNS records have TTL $<$ 300 seconds. Figure 4.7 shows the cumulative distribution of number of domain names per IP address. We observe that 88% of the DNS records only map to one domain name in 300 seconds. We also analyze this in a 1-hour sample of DNS data and observe similar results. Additionally, we analyze the number of IP addresses per domain name in a 300-second period of DNS records. We observe that 35% of the domain names map to more than one IP address. We also analyze this in a 1-hour sample of DNS records and observe similar results. Therefore, we expect our results

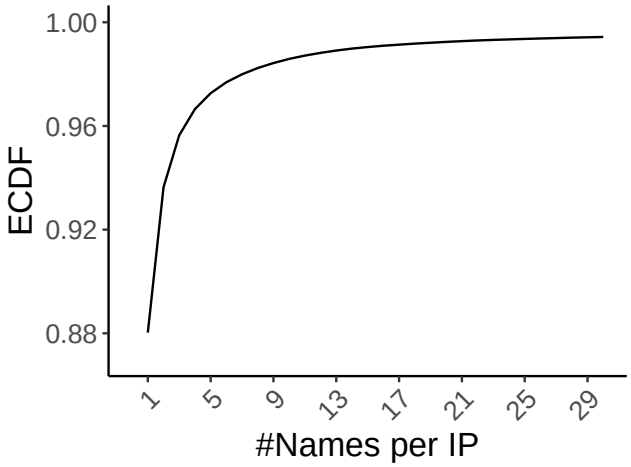


Figure 4.7: Cumulative distribution of number of domain names per IP address

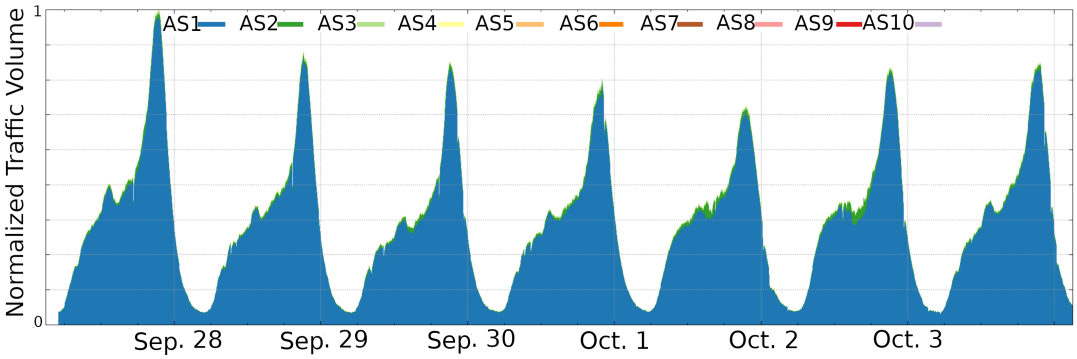
to be accurate for 88% of IP addresses in our flow data. Note that observing multiple IP addresses per domain name, which is a significantly more probable event compared to multiple names for one IP address, does not effect the accuracy of FlowDNS.

4.4 Use Cases

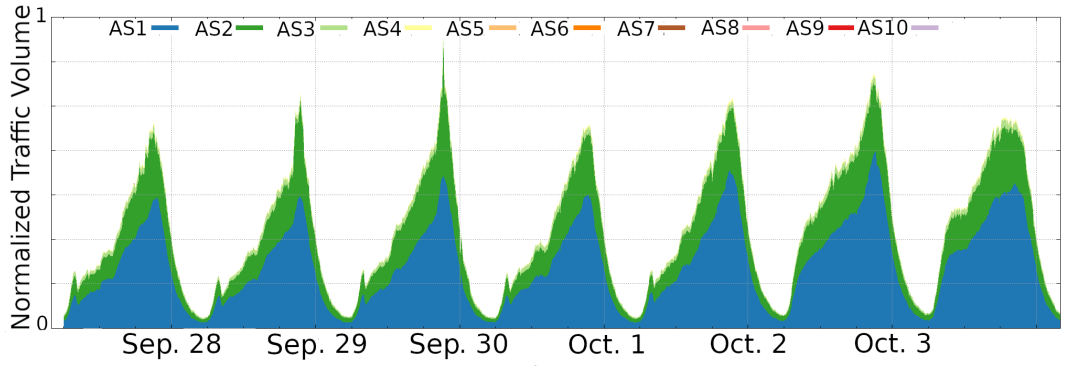
FlowDNS helps ISPs to better plan their networks, while providing the opportunity to analyze the traffic originated by malicious IDN homographs and spam domain names. There have been several studies on detecting malicious or unwanted domain names [139–142], detecting IDN homographs [143–145], and also analyzing domain classification services [146]. However, to the best of our knowledge, there is no work measuring the traffic going to/originated by these domains. In this section, we illustrate three example use cases of FlowDNS, measuring the traffic from malicious or malformed domain names.

For all the following use cases, we use the correlated traffic for over a day in a large European ISP, including 39M unique domain names, and analyze the traffic originated by these domain names.

Network Provisioning and Planning. FlowDNS is already deployed in a large European ISP and a smaller European ISP. The output from FlowDNS is then correlated with BGP data, e.g., source AS, destination AS, hand-over AS, etc., to gain more knowledge about the path the traffic of a specific service takes. Figure 4.8 shows the contribution of different source ASes to the traffic volume of streaming services S1 and S2 over a week at the ISP. As Figure 4.8(a) shows, the traffic corresponding to the streaming service S1 is originated mostly from only one AS, while the streaming service S2 is originated mainly by two ASes as shown in Figure 4.8(b). Note



(a) Streaming service *S1*



(b) Streaming service *S2*

Figure 4.8: Cumulative traffic volume for streaming service *S1* and *S2* per source AS

that AS numbers in two figures do not represent the same ASes necessarily. In Figure 4.8 we observe a diurnal pattern with slight differences between the two services. Knowing the source and intermediate ASes serving a specific service helps ISPs to negotiate with content providers over using ISP's resources instead of a third-party CDN. Also, in case of a broken peering link, it helps find the fallback paths, if they will be overloaded, and which services are effected.

IDN Homographs. Internationalized Domain Names (IDNs) are domain names that contain at least one special non-Latin character. Sometimes, malicious parties abuse non-Latin characters that look similar to Latin characters to deceive users and lead them to their own domain names [147]. We look at the domain names in our correlated traffic to see if any traffic goes to IDN homographs.

To detect actual homographs attacks, we capture all the correlated traffic for a day and filter IDNs. Then, using the Unicode confusable characters list [148], we generate all the permutations of those IDNs to find out whether any well-known domain name is generated out of them. Then, we cross-check the generated permutated domain names with the domain names in our 1-day traffic capture. We do not find any match between the two, probably because the Unicode character list is not comprehensive enough and misses some visually confusable mappings. Therefore, we manually check

all IDNs from the actual traffic, to identify IDN homographs. We specifically find homographs of *google.com*, i.e., googlê.com, góogle.de, góogle.de, etc., and a Cyrillic homograph of *apple.com*.

The amount of traffic going to these IDN homographs is minimal, i.e., only hundreds of megabytes are originated by these homographs in a day. This shows that FlowDNS can be a valuable tool for network operators to identify potentially ongoing phishing campaigns of their users early on. More precisely, 160 MB coming from a homopgraph of *google.com* and 160 MB originated by a homograph of *apple.com*.

Spam Domains. Using our 1-day traffic capture, we check the correlated domain names with the Spamhaus DBL (Domain Block List) [149] to see if any spamming, phishing or otherwise suspicious domains are generating any traffic. To avoid bandwidth limitations on Spamhaus DBL, we sample all the domain names once every hour, giving ca. 1M domain names, out of which 612 are classified as suspicious by the Spamhaus DBL. These include 512 *spam/generic bad reputation domains*, 41 *botnet C&C domains*, 34 *abused spammed redirector domains*, 11 *malware domains*, and 3 *phishing domains*. Collectively, these suspicious domain names originate multiple terabytes of traffic. Figure 4.9 shows a cumulative distribution of traffic volume per number of domain names for each of the above categories. In other words, it shows how many domain names contribute to what fraction of the traffic volume. As can be seen, a significant amount of traffic comes from spam and botnet domains, while only a limited number of domain names account for a large fraction of the traffic.

Malicious websites usually change their domain names rapidly to avoid being detected. Therefore, spam detection datasets such as Spamhaus DBL have an expiry date for their labels, i.e., if checked after the expiry date, they will no longer exist in the dataset and therefore be labeled as benign. FlowDNS allows for real-time checking of the domain names with such datasets.

Invalid Domain Names. RFC 1035 stipulates specifications of DNS domain names [124]. In the following analysis, we focus on three rules to which valid domain names must adhere:

- The total length of the domain name is 255 bytes or less.
- Each label in the domain name is limited to 63 bytes.
- Each label starts with a letter, ends with a letter or digit, and the interior characters are limited to letters, digits, and hyphens.

The word *label* refers to each part of the domain name separated by dots, i.e., if the domain is *A.B.C.com*, labels are: *A*,*B*,*C*, and *com*. In our 1-day traffic capture, we observe that 666k domain names violate at least one of the above-mentioned rules. Figure 4.9 shows that almost all the traffic comes from a very limited number of domain names, and the amount of traffic originated by such domain names is quite significant. Note that the traffic volume is normalized. The most common conflict with the above-mentioned rules is disallowed interior characters. The most common disallowed character found in 87% of the malformatted domains is the underscore character, i.e., "_". Finally, we investigate the overlap of invalid domain names with domains in the spam category and find that only four malformatted domains

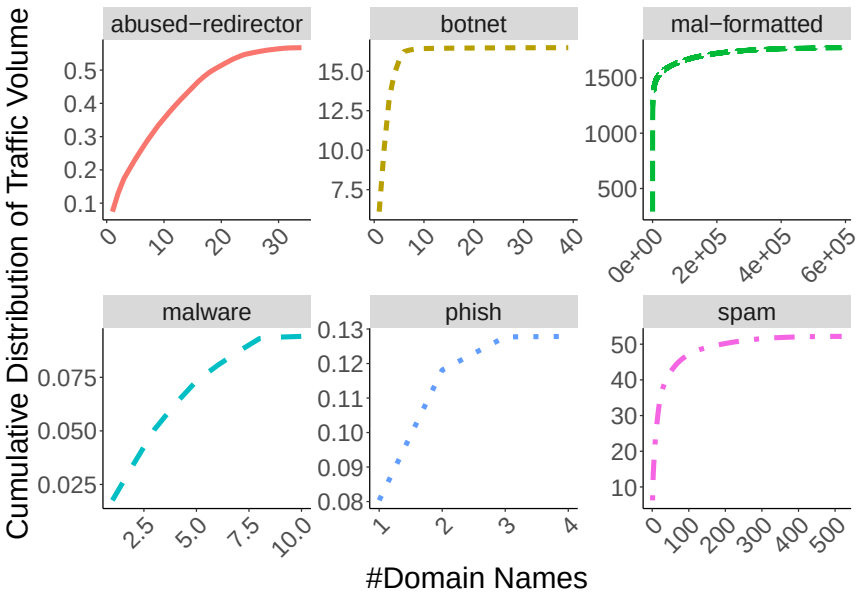


Figure 4.9: Cumulative distribution of the traffic volume per number of domain names

also appear in the spam category. To understand how clients treat these malformed domains, we investigate whether traffic is being exchanged for these domains. We observe that 2.7% of the clients which receive traffic from malformed domains, send traffic back to 23.6% of these malformed domains. This bi-directional traffic accounts for 1.9% of the packets, mostly related to non-web port numbers, e.g., OpenVPN and Kerberos. All other packets are originated by malformed domains and receive no answer.

4.5 Lessons Learned

During the design of FlowDNS, we learned the following lessons:

- When following the CNAME chain, we had to limit the chain length to 6 due to performance reasons. In our experiments, we observed that less than 1% of CNAME chains are longer than 6.
- Splitting the data into several shards allows for higher parallelism, while consuming higher CPU for the same amount of data. Therefore, it is important to keep an eye on this trade-off.
- Buffer rotation, i.e., copying the data once before clearing it, helps to increase correlation percentage without substantial CPU or memory usage in the long run. Therefore, it provides a good trade-off between resource utilization and correlation percentage.
- Expiring DNS records using their exact TTLs induces an unnecessary contention over the shared memory making the loss rate reach over 90%. We tried applying the exact TTLs from the DNS records on our correlation, meaning we correlate

the IP from a DNS record with the source IP from the Netflow record only if the DNS record's TTL plus its timestamp is less than the timestamp from the Netflow record which we consider current timestamp. In other words:

$$TTL_{\text{dns}} + Timestamp_{\text{dns}} < Timestamp_{\text{netflow}}$$

We also run a regular process to clear-up the expired DNS records, when the above-mentioned condition did not hold. We run this on the same sources of data, meaning DNS and Netflow streams at the large European ISP, and observed that the internal buffers of all the streams start to overload from the very first minutes of running the above-mentioned system, with the loss rate of over 90% for both Netflow and DNS streams. We observed that the memory usage reaches up to 45 GB memory usage after only 1 hour of running the system. Comparing this to the results from FlowDNS in Figure 4.4(b), we observe that the memory usage is doubled although only 10% of the data is received at the system and others are lost. This could be due to the regular clear-up process not being fast enough to clear-up all the expired TTLs as the hashmaps grow, while at the same time, the contention to access the shared memory is so high that the performance degrades dramatically.
Therefore, we learned that using rotating buffers with a common expiry time instead of the exact value helps in gaining the same correlation rate compared to keeping the DNS records forever, with no loss and is much more resource-efficient.

We hope that these lessons will prove useful for fellow network application developers and researchers alike.

4.6 Summary

Inferring the services behind a certain traffic flow is not possible merely by looking at the IP addresses due to the prevalent deployment of CDNs. In this chapter, we presented FlowDNS, a system to correlate DNS and Netflow streams in real-time. We used several techniques such as splitting the data, rotating buffers, and specific hashmaps to keep track of longer-living DNS records. We evaluated each of these techniques and confirmed the usefulness of each. Then, using FlowDNS, we analyzed the domain names with known datasets to detect malicious domains and observed that a substantial amount of traffic is originated by these domain names. Moreover, we checked the adherence of those domain names to standardization rules and observed that 1.7% of all the domain names violate them. We also found that the traffic originated by such domains accounts for 0.5% of the daily traffic. Finally, we make FlowDNS available to fellow researchers and network operators.

Now that we discussed the systems to monitor network traffic in Chapter 3 and to correlate them with DNS information to gain more insights about the actual application behind the traffic in this Chapter, we plan to move towards finding hidden characteristics of the Internet traffic, and investigate the reasons a specific abnormal Internet traffic exists in Chapter 5.

5

Zeroing in on Port 0 Traffic in the Wild

Transport protocols use port numbers to identify different Internet services. There are different categories of port numbers: Officially registered ports at IANA [11], unofficially but well-known ports, and dynamic ports, which cannot be registered and are free to use by anyone. In contrast, there are also some ports which are reserved and should not be used. One of these reserved port numbers is port 0. It is reserved in most common transport layer protocols, i.e., TCP [150], UDP [150], UDP-Lite [151], and SCTP [152]. When providing a port number 0 to the `bind()` system call to establish a connection, operating systems generally choose a free port from the dynamic range [153, 154]. Therefore, one needs to create a raw socket in order to send port 0 packets.

In Chapter 3, we proposed a system to monitor network traffic to find about traffic characteristics. Using Flowyager, we uncovered a non-negligible share of traffic using port number 0. Previous work has also confirmed such a phenomenon both in darknets and the Internet [5, 10, 155].

In this chapter, we shed light on port 0 traffic in the Internet, by analyzing the traffic from real networks, rather than darknets as is done in most related work, and by performing active measurements to survey the real-world reaction of hosts and routers to port 0 traffic.

To the best of our knowledge, this is the first work which conducts both active and passive measurements on port 0 in the Internet, to better understand port 0 traffic characteristics and origins. Specifically, this work has the following three main contributions:

- We leverage a flow-level dataset from a large European IXP to inspect the origins of port 0 traffic (cf. Section 5.3). We find that out of the top 10 ASes originating port 0 traffic, the majority does not follow typical diurnal patterns of common protocols such as TCP/80.
- We inspect four packet-level datasets to discover the actual contents and detailed characteristics of port 0 packets (cf. Section 5.4). We show that the majority of non-empty packets in UDP are related to BitTorrent. We find that most TCP packets do not contain any payload and are one-way. However, most of the two-way TCP streams are scanning artifacts.
- We perform active measurements both in IPv4 and IPv6 to gain a tangible perspective over port 0 responsive IP addresses (cf. Section 5.5). We find that

IPv4 traffic using TCP uncovers a substantial number of responsive hosts in a small number of ASes. We also perform traceroute-style active measurements to better understand port 0 traffic filtering in wild, and find discrepancies between IPv4 and IPv6. Finally, we will run periodic port 0 measurements and make the results available to the research community.

5.1 Related Work

Already in 1983, Reynolds and Postel specified that port number 0 is reserved in TCP and UDP [150]. Over the course of several years, similar provisions have been introduced for other transport protocols as well [151, 152]. Traffic sent from or to port 0 thus violates these specifications. Fittingly, most reports on port 0 traffic are associated with DDoS attacks [156–158] and malformed packets [159].

Even though there is traffic on port 0 in the Internet, there is little research on its root causes. Motivated by port 0 traffic spikes observed in November 2013 at the Internet Storm Center and reports from security researchers at Cisco Systems, Bou-Harb et al. [155] study port 0 traffic on 30 GB of darknet data. They filter out any misconfigured traffic and packets with non-conforming TCP flags common in backscatter traffic [160]. Using fingerprinting techniques [9], they argued that more than 97% of their identified port 0 traffic was related to probing activities, some orchestrated by malware.

In 2019, Luchs and Doerr [10] revisit the case of port 0 traffic, by studying data obtained from a /15 darknet over a period of three years. They find that out of about 33 000 source IP addresses involved in port 0 traffic, 10% can be attributed to DDoS attacks, 6% to OS fingerprinting, and less than 1% to scanning activities. When aggregating by the number of packets instead, scanning traffic dominates with 48% of all port 0 packets.

More recently, Maghsoudlou et al. [5] analyze port 0 traffic for a single passive measurement source. Similarly to our results, they find that a small number of ASes are responsible for about half of all port 0 traffic.

In contrast to the related work [5, 10, 155], which all focus their efforts on the analysis of a single passive data source, in this chapter, we analyze four complementing passive datasets in addition to conducting an active measurement campaign to better understand port 0 traffic in the wild.

5.2 Datasets Overview

We leverage two different kinds of passive datasets to study port 0 traffic characteristics: Flow-level and packet-level data. Throughout the paper, port 0 traffic refers to the subset of the traffic which has either source port or destination port or both set to zero. Flow-level data gives us a high-level overview of Internet traffic and

Dataset	IXP-2020-01	MAWI-2006-2020	MAWI-2020-04	Waikato-2011-04	CAIDA-2019-01
Timespan	Jan. 25–31, 2020	2006–2020	Apr. 8–9, 2020	Apr.–Nov., 2011	Jan. 17, 2019
Duration	1 week	14 years	2 days	86 Days	2 hours
Format	Flows	Packets	Packets	Packets	Packets
%IPv4,IPv6 (Port0)	99.8%,0.2%	100%,0%	100%,0%	100%,0%	99.7%,0.3%
%UDP,TCP (Port0)	96.8%,3.2%	22.4%,77.6%	30.2%,69.8%	15.5%,84.5%	43.8%,56.2%
Payload	No	Yes	Yes	Yes	No
Sampled	Packet-based	Time-based	No	No	No
# Packets	34.3×10^9	23×10^9	15.9×10^9	27.822×10^9	8.2×10^9
% Port 0 packets	0.25	0.0008	0.0001	0.002	0.0002
# Bytes	25.5 TB	14.6 TB	6.7 TB	16.9 TB	4.3 TB
% Port 0 bytes	0.28	0.00012	0.0002	0.001	0.00002

Table 5.1: Overview of passive port 0 datasets

can be used to analyze the aggregate flow of traffic. In our case, we use one week of IPFIX flow data from a large European IXP. On the other hand, to be able to dissect detailed traffic characteristics like fragmentation, header flags, and different payloads, we need to inspect every single packet. Therefore, we use four different packet-level datasets, namely a long-term and a short-term MAWI dataset, CAIDA, and Waikato. Different packet-level datasets are used to cover different geographical and temporal vantage points.

As shown in Table 5.1, we use the following datasets:

IXP-2020-01 One week of sampled IPFIX data from the end of January 2020 captured at a large European IXP.

MAWI-2006-2020, MAWI-2020-04 These datasets [107] contain packet traces from the transit link of the WIDE backbone [161] to the upstream ISP captured at samplepoint-F. They include partial packet payload. To obtain a more comprehensive view, we use two variants of MAWI datasets:

MAWI-2006-2020 This dataset captures 15-minute snapshots each month from January 2007 to July 2020.

MAWI-2020-04 We also use the most recent MAWI dataset being part of the Day in the Life of the Internet project [162], which is April 8–9, 2020.

CAIDA-2019-01 This dataset [163] contains anonymized packet traces without payload from CAIDA's passive monitors. For our analysis we use the most recent dataset available at the time of writing, which is the one-hour period from 14:00–15:00 UTC recorded on January 17, 2019.

Waikato-2011-04 This dataset [164] contains packet header traces including the first few bytes of payload and is captured at the border of the University of Waikato network in New Zealand.

We analyze port 0 traffic seen in passive data in detail in Sections 5.3 and 5.4. In addition to passive flow and packet data, we also conduct active measurements. More specifically, we run two types of measurements to analyze responsiveness on port 0 and filtering of port 0 traffic in the Internet:

Port scan We use ZMap [19, 26] and ZMapv6 [20] to find responsive addresses on port 0. In IPv4 we conduct Internet-wide measurements, in IPv6 we leverage an IPv6 hitlist [23, 165, 166].

Traceroute We use Yarrp [22, 167] to traceroute addresses in IPv4 and IPv6 prefixes in order to analyze port 0 traffic filtering in the Internet.

We present results from our active measurement campaign in Section 5.5. By leveraging both passive and active measurements we can analyze different aspects of port 0 traffic in the wild.

5.2.1 Ethical Considerations

We followed best current practices as explained in Section 2.3 for both our passive and active measurement. Contrary to the active measurements, we will not publish any passive measurement data.

5.2.2 Reproducible Research

To foster reproducibility in measurement research [168, 169], we make data, source code, and analysis tools of our active measurements publicly available [170]. Due to privacy reasons we will not publish data from the passive datasets.

5.2.3 Continuous Port 0 Measurements

To allow further analysis of port 0 responsiveness and filtering over time, we periodically run active port 0 measurements. The raw results of these measurements are publicly available for fellow researchers at:

inet-port0.mpi-inf.mpg.de

5.3 Flow-level Analysis

Analyzing the traffic flowing between different Autonomous Systems is helpful to detect high-level patterns. To investigate port 0 traffic patterns, we use the IXP-2020-01 dataset and inspect the ASes originating or being targeted by port 0 traffic. In one week of IXP flow data, we find 23 000 ASes contributing to port 0 traffic. We observe that the source AS with highest number of packets in sends port 0 traffic to 4357 distinct destination ASes. Also, the destination AS with highest number of port 0 packets being destined to, is targeted by 1245 distinct source ASes.

We also observe that in 9 out of 10 top source ASes involved in port 0 traffic, port number 0 is among the top-5 source and destination port numbers along with TCP/80 (HTTP) and TCP/443 (HTTPS). We find that more than 99% of port 0 traffic has

both source and destination port set to zero. Interestingly, more than 99% of all TCP traffic contains no TCP flags. This leads us to believe that this is not actual port 0 traffic and is most likely an artifact of packet fragmentation [171], which is incorrectly classified as TCP/0 traffic by the flow exporter [172]. We also analyze the 1% of the TCP traffic with non-zero TCP flags, composed of 867 packets. We find that 30% of this traffic sets their TCP flags to CWR/URG/ACK, 27% to ACK only, and 25% to URG/ACK/PSH/SYN. 62% of this traffic has an average packet size of less than 100 bytes, while 18% has an average packet size of more than 1480 bytes. To investigate more in-depth on how different networks react to port 0 traffic, we perform active measurements (cf. Section 5.5).

To further investigate origins and causes of port 0 traffic, we analyze the diurnal patterns of traffic originated by the top 10 source ASes and compare them with the more common Web traffic on TCP/80. Figure 5.2 shows the hourly patterns of port 0 traffic grouped by source AS, compared with the total TCP port 80 traffic as a reference for regular traffic. Weekends are highlighted with a yellow background. Figure 5.1 shows a heatmap of the Spearman correlation of the diurnal patterns of these ASes and TCP/80 traffic. We see that while AS2 is the most correlated to TCP/80 traffic, AS4 and AS7 show highly similar patterns to each other and moderate correlation to TCP/80 traffic. Moreover, AS3 shows a unique pattern with no correlation to either other ASes or TCP/80.
AS3 is a cloud computing provider while other ASes are web hosting providers, ISPs, or telecommunication companies. The unique traffic pattern originated by AS3 implies irregular usage such as scanning or reset attack.

To better understand the causes of port 0 traffic, we analyze average payload sizes observed in the IXP-2020-01 dataset. For easier comparison with the packet-level datasets (cf. Section 5.4), we choose to analyze the payload size instead of the average packet size reported directly in the flow data. We estimate the payload size by subtracting the IP and TCP/UDP headers without options. As shown in Figure 5.3, for TCP, we observe that nearly 88% of packets are smaller than 100 bytes, while in UDP, more than 75% of packets are larger than 100 bytes. Having roughly 20% full-sized packets in UDP, along with many mid-sized packets, indicates possible fragmentation. Unfortunately, our IPFIX dataset does not include fragmentation information for IPv4 flows. It does, however, include information about the IPv6 next header value. We find no IPv6 flows with the next header value set to fragmentation (i.e., 44). To investigate further on the exact fragmentation header flag, we inspect the IPFIX field containing a list of all IPv6 extension headers in a flow. We find, however, that the content of this IPFIX field does not conform to the IPFIX specifications as defined by the RFC. This is possibly due to an erroneous early version of the RFC, which has since been corrected [173]. As IPFIX datasets usually depend heavily on how their exporter is implemented, researchers who would like to work on them should be extra cautious to make sure that their data is flawless.

To summarize, multiple indicators lead us to believe that most of port 0 traffic seen at the IXP is an artifact of packet fragmentation. Nevertheless, we find that the IXP data gives valuable information on diurnal patterns. By analyzing the correlation

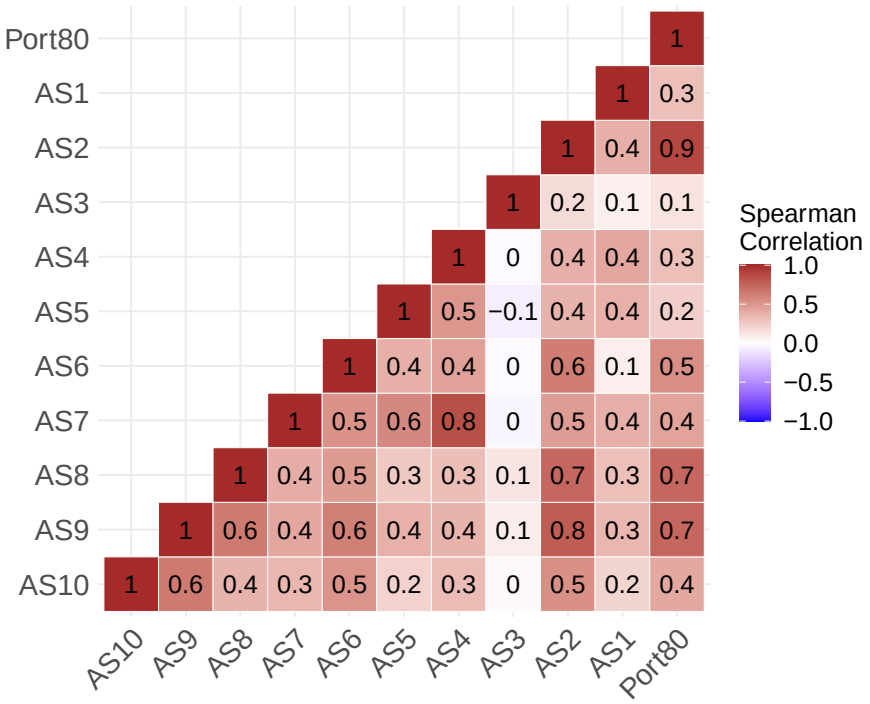


Figure 5.1: Correlation coefficients between port 80 traffic and the top 10 source ASes involved in port 0 traffic in the IXP

between diurnal patterns of different ASes and port 80 traffic, we find one AS deviating heavily from the common diurnal patterns. This indicates possible scanning or other irregular activities which requires a more in-depth analysis which can only be performed on packet-level data. Therefore, we analyze the four packet-level datasets in the upcoming section.

5.4 Packet-level Analysis

Although using a flow-level dataset provides us with useful information about the origin and targets of port 0 traffic, it cannot provide information on what the packets actually contain. Knowing the packet content, we can infer the cause of port 0 usage more precisely. To this end, we use the MAWI-2006-2020, MAWI-2020-04, CAIDA-2019-01, and Waikato-2011-04 datasets. CAIDA-2019-01 contains no payload, while others provide partial payload data. We begin our packet-level analysis by investigating packet payload sizes, for which we use the packet length field found in UDP and TCP headers. As Figure 5.3 shows, nearly all packets in MAWI-2020-04, MAWI-2006-2020 and Waikato-2011-04 have a payload size of less than 100 bytes. In both the MAWI-2020-04 and the CAIDA-2019-01 dataset, more than 99% of the TCP port 0 traffic does not have any payload, while UDP traffic always contains payload. Note that Figure 5.3 only shows those TCP packets with payload, i.e., for CAIDA-2019-01 and MAWI-2020-04, it shows less than 1% of all TCP packets. In the CAIDA-2019-01 dataset, while UDP traffic includes payload sizes smaller than 104 bytes in 99% of the

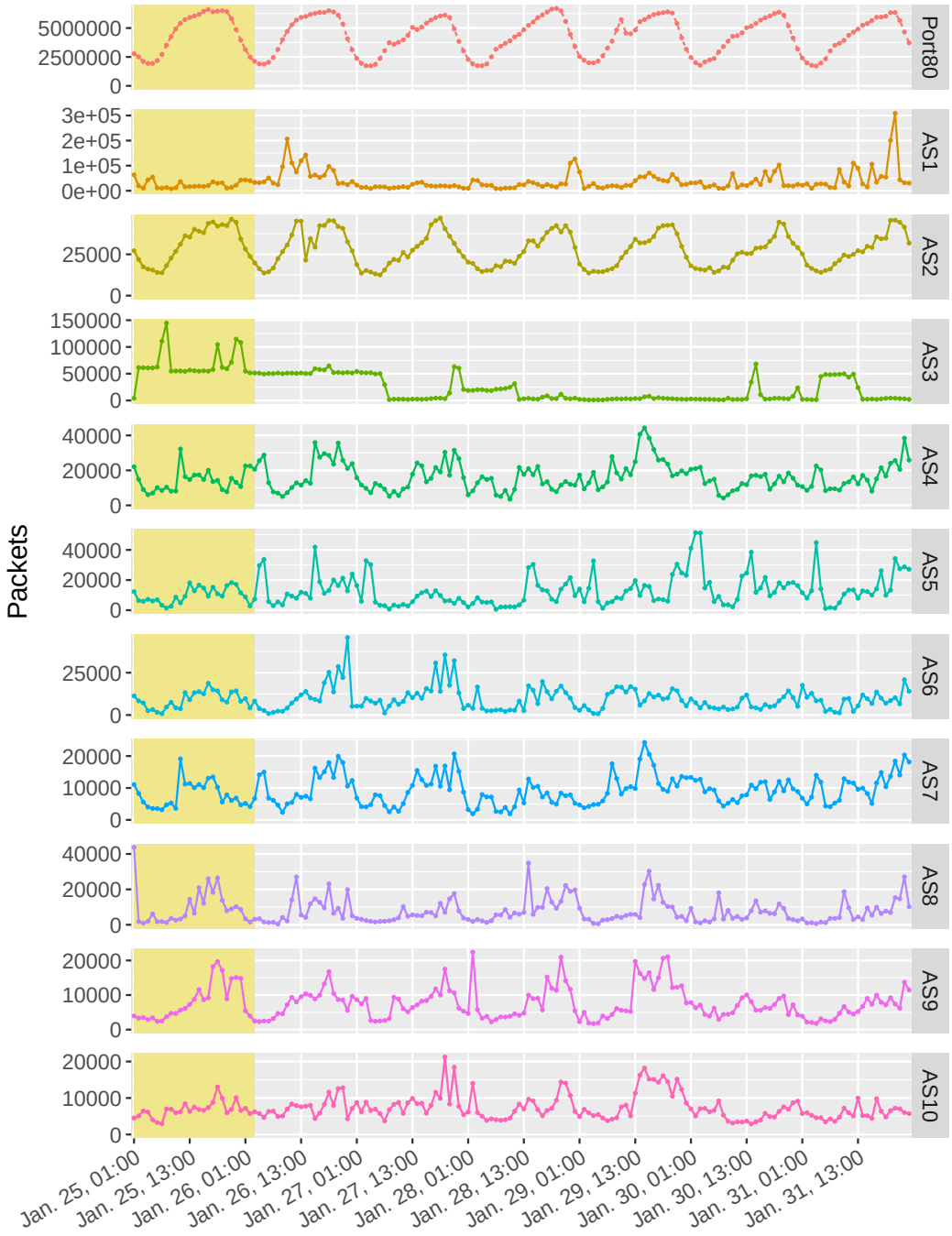


Figure 5.2: Daily traffic pattern of port 80 and top 10 source ASes for port 0 in IXP-2020-01 dataset in IPv4

packets, TCP traffic shows more mid-sized payload sizes. Investigating further into the CAIDA-2019-01 dataset shows that all packets contain zero as fragment offset and all the fragmentation flags are set to *Don't Fragment*. This suggests that port 0 traffic in the CAIDA-2019-01 dataset is likely not a fragmentation artifact. However, we find some bogus packets, e.g., with zero header length among these mid-sized TCP packets.

Similar to our analysis in Section 5.3, we investigate port 0 traffic origins and destinations in our two MAWI datasets. We find that most of the traffic, namely more than 60%, is destined to only 2 ASes, as shown in Figure 5.4. Figure 5.5 shows the cumulative distribution of IP addresses in port 0 traffic in different datasets. We exclude the MAWI-2006-2020 dataset since aggregating through 14 years would not give us useful information. We observe that more than 75% of port 0 traffic is originated by less than 10 IP addresses in CAIDA-2019-01, IXP-2020-01, and MAWI-2020-04. Also in all the datasets, more than 87% of port 0 traffic is destined to less than 10 IP addresses.

In Figure 5.6, we show the payload distribution classified with libprotoident [174] for each year in the MAWI-2006-2020 dataset. The red line along with the right Y-axis show total number of packets throughout different years. The stacked bar plots show different categories of payloads excluding *No Payload* and *Unknown UDP*. We find that BitTorrent traffic is a constant contributor to port 0 traffic in Waikato-2011-04, MAWI-2020-04, and in different years in MAWI-2006-2020.

In MAWI-2020-04, we find that 70% of the payloads belong to the BitTorrent UDP protocol. Additionally, a payload pattern covering 16% of the traffic, probably belonging to a custom application-layer protocol, DNS, OpenVPN, and NTP, contributes to other payloads in MAWI-2020-04 port 0 traffic in our dataset. In Waikato-2011-04, BitTorrent-UDP and Skype are among the top payloads.

In MAWI-2020-04, MAWI-2006-2020, and CAIDA-2019-01, Malformed packets contribute less than 2% to port 0 packets, e.g., with wrong checksums, having UDP length of higher than IP length, etc. However, in Waikato-2011-04, we find that 16.2% of the traffic is malformed. This shows that port 0 traffic can also be caused

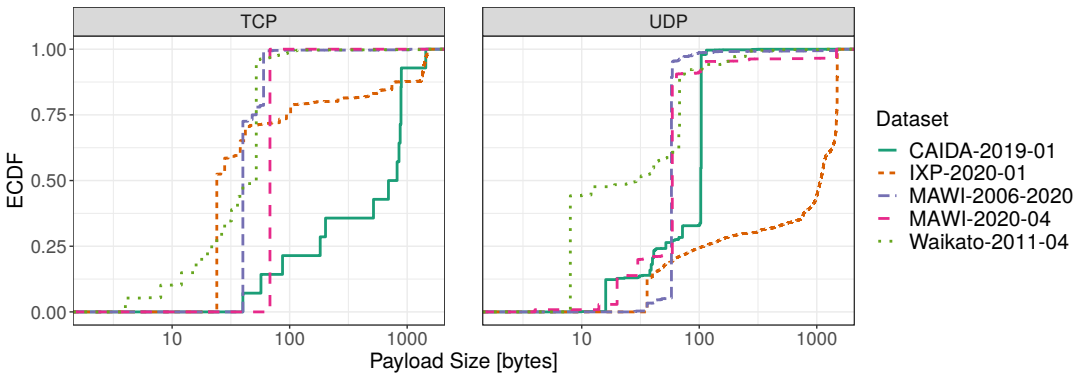


Figure 5.3: Cumulative distribution of payload size in port 0 traffic. Note that the X-axis is log-scaled.

by misconfiguration, programming errors, or people sending malformed traffic on purpose.

Next, we analyze different TCP flags in packet-level datasets to better understand possible causes of port 0 traffic. Attackers and scanners usually use specific TCP control bits in their packets to achieve their goals. For instance, attackers sending spoofed traffic set the SYN bit to try to initiate TCP connections with their targets, which in backscatter traffic we see as SYN/ACK, RST, RST/ACK, or ACK packets [160]. Therefore, we investigate TCP control flags in the datasets. We observe that most of the TCP flags are only SYNs: More than 66% in MAWI-2020-04, and 92% in CAIDA-2019-01, which might indicate that most of the TCP port 0 traffic in these two datasets is caused by scanning. We analyze TCP flags in MAWI-2006-2020 dataset per year, as shown in Figure 5.7. First, we check whether all packets in a TCP stream are one-way or two-way. We find that a large fraction of the TCP streams are one-way. This also holds for all other packet-level datasets. Then, we categorize two-way TCP streams as follows:

- Scan to closed port: Client sends SYN, receives RST or RST/ACK.
- Scan to open port: Client sends SYN, receives SYN/ACK, client then sends RST or RST/ACK.
- No SYN: No SYN is ever sent. The stream begins with other flags, mostly SYN/ACKs followed by RSTs from the other side.
- Not scan: None of the above, i.e., client sends SYN but receives no RST.

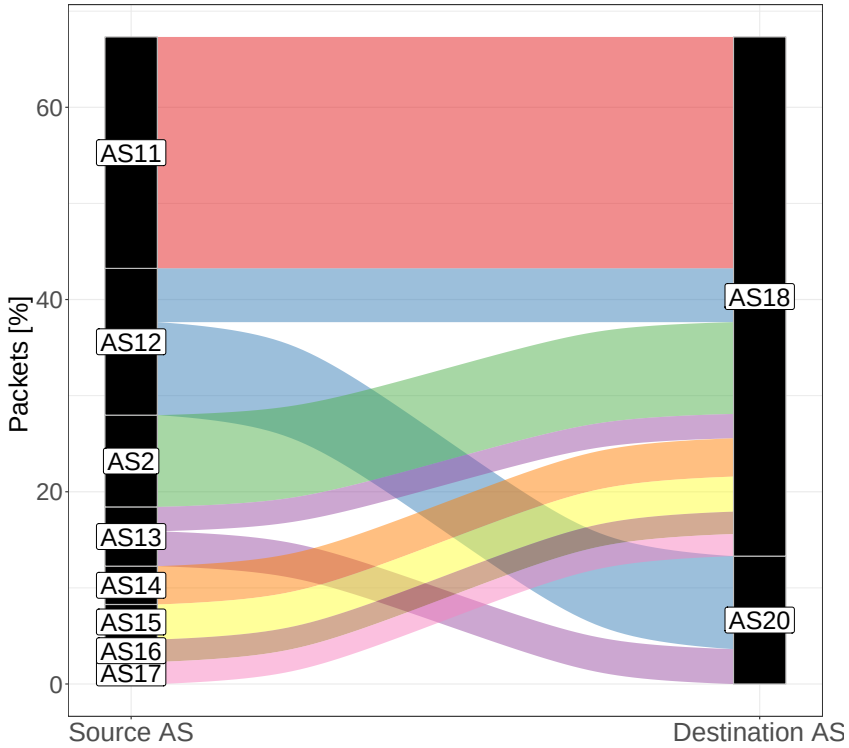


Figure 5.4: Traffic between top 10 (source AS, destination AS) pairs involved in port 0 traffic in the MAWI-2020-04 dataset

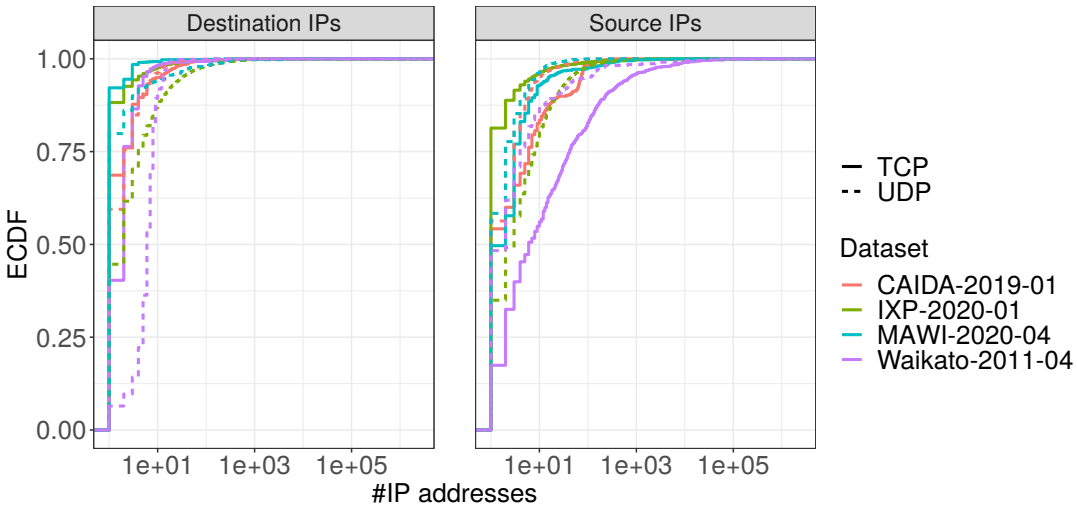


Figure 5.5: Cumulative distribution of IP addresses in port 0 traffic. Note that the X-axis is log-scaled.

We find that a major fraction of two-way TCP streams are scans to closed ports for most of the years. Among the streams in the *Not scan* category, we find two long streams of ACK/PSH followed by multiple ACKs in 2015 or ACK/PSH/FIN in 2019, respectively. We believe that these streams are related to an ACK/PSH flood attack [175] considering the relatively high number of packets sent in these streams. Next, we analyze specific years of the MAWI-2006-2020 dataset with very characteristic spikes more in-depth. In 2009, we see the largest number of total packets of any year, with a TCP:UDP ratio of about 2:1. The majority of UDP traffic is originating with source port UDP/8000 from many different IP addresses within a Chinese ISP AS which are mostly destined to UDP/0 towards a single IP address belonging to a Japanese university inside WIDE. For TCP, the majority of traffic is sourced from a single IP address within a Canadian ISP and destined to many different IP addresses. Almost all sources are TCP/0 and the destinations are TCP/22 (SSH). As is shown in Figure 5.7, these are very likely scanning activities.

In 2012 we see the largest number of TCP streams as shown in Figure 5.7. We find a factor of 54 times more TCP traffic this year than UDP traffic. Almost 80 % of all TCP/0 traffic is from a single IP address within a hosting company, the destination addresses and ports are evenly distributed. The TCP flags of all packets are set to RST/ACK. These indicators lead us to believe that this is backscatter traffic from attack traffic using spoofed IP addresses [160].

Finally, we investigate the current year 2020, from January to July. During this period we see 26 times as much TCP traffic compared to UDP. The majority of TCP traffic originates from a single IP address at a hosting company, which uses TCP/43573 as a source port. For the IP address in question we find many different reports on abuse DB websites, which hint at scanning and vulnerability probing.

To summarize, we find that a large fraction of TCP streams in port 0 traffic is one-way. However, we still see some two-way streams related to scanning activities. Analyzing

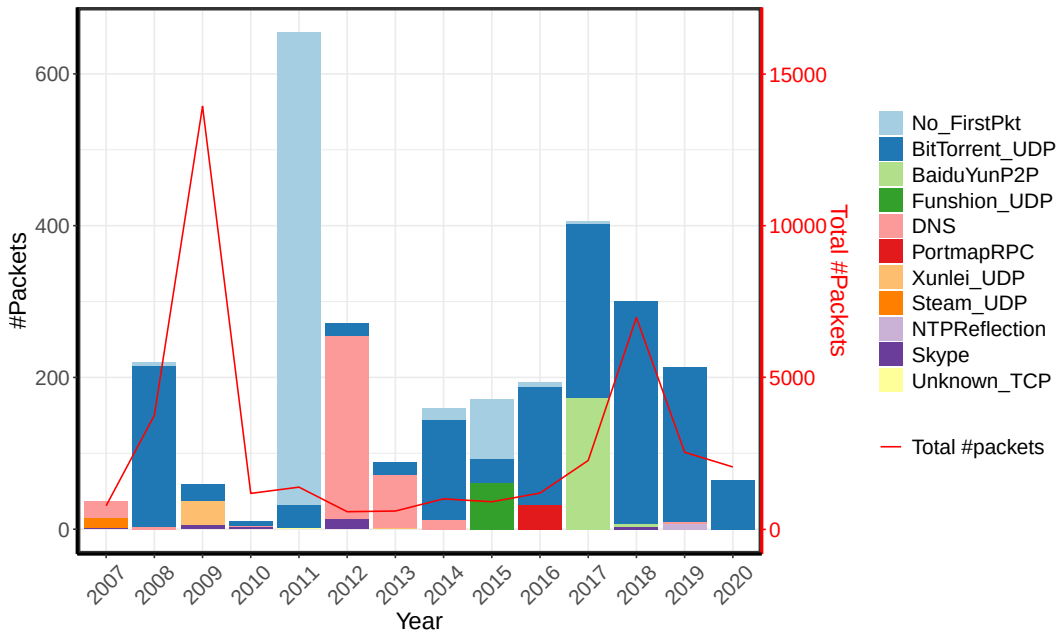


Figure 5.6: Payload distribution (bar plots) and total packet count (red line) for MAWI-2006-2020

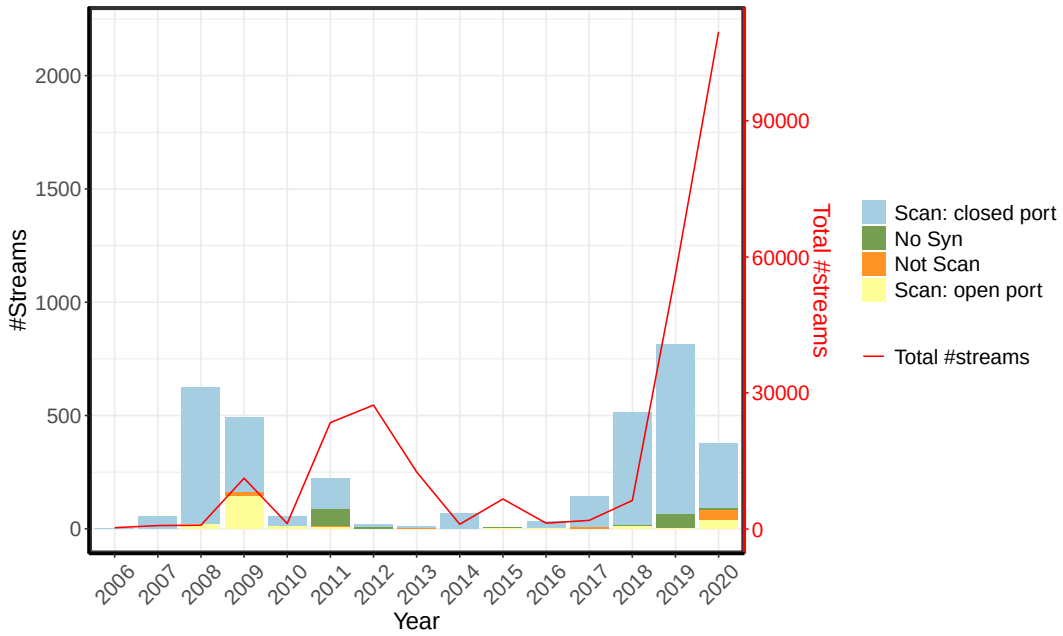


Figure 5.7: TCP stream categorization (bar plots) and total streams (red line) for MAWI-2006-2020

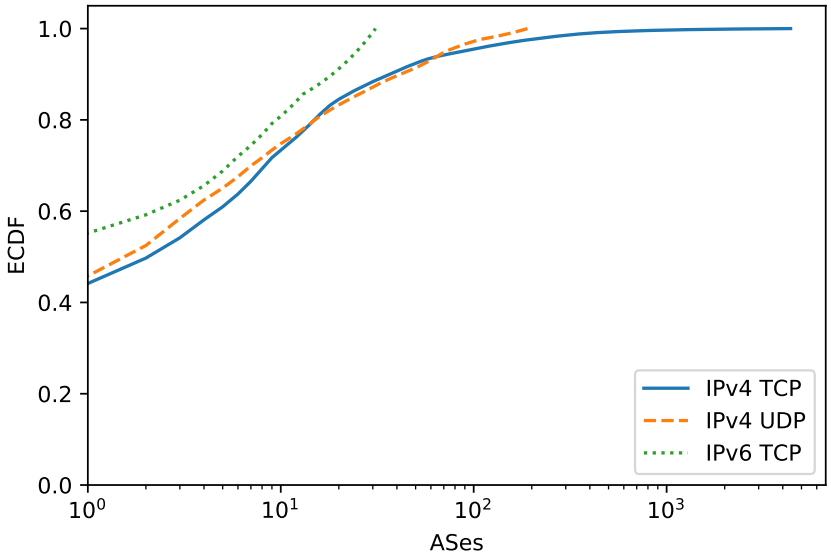


Figure 5.8: Cumulative distribution of responsive IP addresses per AS. Note that the X-axis is log-scaled.

packet payloads throughout all our datasets, we observe that BitTorrent UDP traffic is a constant contributor to port 0 traffic.

5.5 Active Measurements

As discussed in the previous sections, we observed a significant number of RST/ACKs and even some SYN/ACKs which indicate scanning activities. To better understand how the network reacts to port 0 traffic, we stage an active measurement campaign. We run two types of measurements: (1) Port scan measurements allow us to analyze responsiveness of IP addresses to port 0 probes and (2) traceroute measurements provide information on where port 0 packets are being filtered.

5.5.1 Responsive Addresses

We run four types of port scan measurements, for each possible combination of IPv4/IPv6 and TCP/UDP. The IPv4 measurements are run on the complete address space minus a blocklist, the IPv6 measurements use an IPv6 hitlist [166]. For the TCP measurements we send regular SYN packets, for UDP we send the most prominent payload found in our passive packet traces.

For the four protocol combinations, we get vastly differing results. With 2.3 M, the largest number of addresses responds to our IPv4 TCP port 0 probes. Only 2222 unique addresses respond to IPv4 UDP probes and 120 respond to IPv6 TCP probes. We find not a single responsive address for IPv6 UDP probes.

	ASN	AS Name	Count
1	6830	Liberty Global	4822
2	6327	Shaw	3257
3	812	Rogers	2297
4	33915	Vodafone	2152
5	11492	Cable One	1095
6	30036	Mediacom	688
7	12389	Rostelecom	643
8	4134	Chinanet	575
9	3320	Deutsche Telekom	552
10	4766	Korea Telecom	498

Table 5.2: Top 10 ASes of non-reachable target addresses when comparing TCP/0 and TCP/80

When mapping responsive addresses to ASes [176, 177], we find that a small number of ASes makes up the majority of responses. Figure 5.8 shows the AS distribution of responses for the different protocols. The top ten ASes make up 72 %, 73 %, and 79 % of all responses for IPv4 TCP, IPv4 UDP, and IPv6 TCP, respectively. When we look at the overlap of responding addresses in TCP and UDP for IPv4, we find that 61 % of IPv4 UDP addresses are present in IPv4 TCP results. In IPv4 TCP, where we see the most responses by far, most of the top 10 ASes belong to ISPs. This leads us to believe that faulty or misconfigured ISP equipment is to blame for responses to port 0 probes.

Next, we analyze the initial TTL (iTTL) value [178–180], UDP reply payload, and combine these with the responding AS. For IPv4 TCP we find that the most common iTTL values are 64 (57 %), which is the default for Linux and macOS, 255 (36 %), the default for many Unix devices, and 128 (7 %) the default for Windows. When combining these iTTL values with the responding AS we find no clear patterns. In contrast, for IPv4 UDP we find a clear correlation between iTTL, payload, and AS. The most common response payload (32 %) is sent from six different ASes with an iTTL value of 32 or 64. The second most common response payload (14 %) is identical to our request payload, i.e., the probed hosts simply mirror the payload that they receive. Packets with this payload originate from a single AS (AS7922, Comcast) and all of them have an iTTL of 255. The third most common payload (8 %) is made up of 16 zero bytes and originates from AS14745 (Internap Corporation) with an iTTL of 32.

These findings suggest that only a small number of networks contain misconfigured devices erroneously responding to port 0 probes.

5.5.2 Port 0 Traceroutes

To better understand how port 0 traffic is handled inside the network, we conduct traceroute-style measurements using Yarrp [22]. This allows us to see if port 0 traffic is treated differently by routers compared to standard TCP/80 or TCP/443 traffic.[1] In IPv4, we split the announced address space into 11 M /24 prefixes and send a trace to a random address within each of these prefixes. In IPv6, the equivalent would be sending traces to every /48 prefix. This is, however, not feasible due to the vast address space. Therefore we decide to pick one random address per announced IPv6 prefix, no matter the prefix length. We ensure that random addresses for less specific prefixes do not fall into more specific prefixes. In total, we send probes to about 88 k IPv6 prefixes.

5.5.2.1 Reachability

When analyzing the reached target addresses depending on the used port numbers, we find that in IPv4 there is a significant difference between port 0 and other ports. 91 k of IPv4 port 0 traces reach their target, whereas 118 k traces on TCP/80 and TCP/443 reach their target IPv4 address, an increase of almost 30 %.

In IPv6, however, almost no targets are reached for either port number, as the likelihood of a randomly generated address in a prefix actually being assigned is quite low. Therefore, we perform additional analyses based on the reachability of the target BGP-announced prefix.

The general picture in IPv4 does not change drastically when analyzing the reachability of the target prefix: Port 0 probes reach fewer target prefixes compared to port 80 and port 443 probes, although the difference is reduced to 14.2 % and 9.5 %, respectively.

When we analyze the reached target prefixes for IPv6, however, we see a slight difference of 3 %.

As the difference of reachable addresses is most apparent in IPv4, we investigate this phenomenon in more detail. We identify on a per-target basis the addresses which see no responses in TCP/0, but do see responses in TCP/80. These non-responsive port 0 addresses are mapped to 4102 distinct ASes, exhibiting a long-tailed distribution. Next, we check whether we find other addresses in these 4102 ASes to be responsive to port 0 traceroutes, to exclude the possibility of missing responses due to ICMP rate limiting. We find responses to port 0 traceroutes for only 15 of these ASes, making up only 0.4 % of the total 4102 ASes. This underlines the fact that these ASes are indeed handling port 0 traceroutes differently compared to other ports. Furthermore, as is shown in Table 5.2, 9 out of the top 10 ASes belong to ISPs, further indicating that these might be ASes blocking port 0 traffic to their clients[181–183]. In the rest of

[1] Note that due to the nature of traceroute measurements, missing traceroute responses could stem either from filtered packets on the forward path, rate-limiting of ICMP packets at the routers, as well as dropping of ICMP responses on the return path.

this section, we analyze many additional aspects of traceroute responses, by checking for differences in the last responsive hop, comparing the number of responsive hops per trace, evaluating ICMP types and codes.

To summarize, our findings show that packets are handled differently based on the destination port number. Port 0 is more likely to be filtered on the path as well as at the target hosts. Interestingly, the phenomenon of fewer responses for TCP/0 seems to be much more common in IPv4 compared to IPv6, which could be due to inconsistent firewall rules [184].

5.5.2.2 Last Responsive Hops

We analyze the last responsive hop of each trace specifically. More concretely, we are interested in the distance, i.e., the largest TTL value of traceroutes, where we get an ICMP response to. This allows us to determine whether TCP/0 traceroutes are e.g. dropped earlier in the network and therefore are terminated earlier in the Internet.

Therefore, we compare the distribution of the last responsive hop. The left part of Figure 5.9 shows the distribution of the last responsive hop for IPv4 and IPv6, respectively. The only visible difference we see for IPv4 are the lower whiskers for TCP/0, stemming from the fact that TCP/80 and TCP/443 has slightly more outliers with high TTLs when it comes to the last responsive hops. For IPv6 we see that TCP/0 has a median of 13 and TCP/80 as well as TCP/443 have a median last responsive hop TTL of 14. Since the median is almost identical, this is due to the median only being able to represent integer values if all elements (namely path lengths) are integers. TCP/0's median is therefore "just below" 14 and the others' median is "just above" 14. All in all, the box plots show that there is no significant difference when analyzing last responsive hops depending on the transport port.

5.5.2.3 Number of Responsive Hops

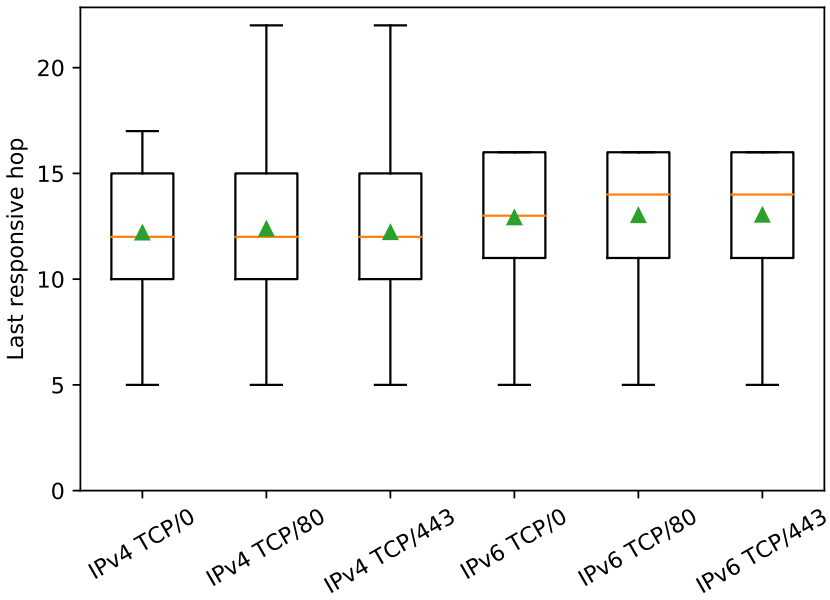
Next, we try to answer the question whether fewer routers on the path send ICMP messages for port 0 traceroute traffic or not.

In the right part of Figure 5.9 we show the box plot of the number of responsive hops. Again, we see no evidence of router sending fewer ICMP responses for port 0 traffic. We see a slight reduction of TCP/443 ICMP responses per trace in IPv4.

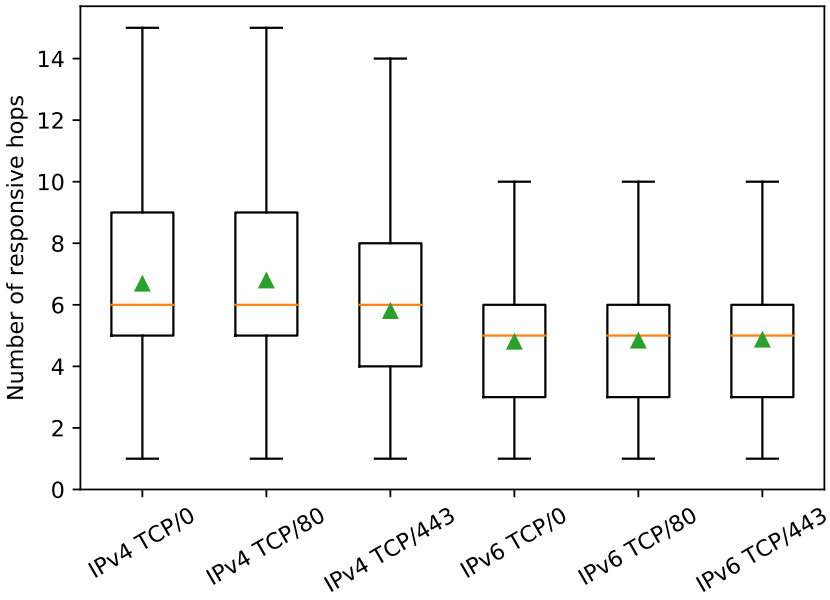
5.5.2.4 ICMP Types and Codes

Finally, we evaluate the different ICMP types and codes sent by routers.

Figure 5.10 shows the distribution of type and code combinations for ICMP and ICMPv6, respectively. As expected, the vast majority are of type "Time to Live exceeded in Transit" for IPv4 and 'hop limit exceeded in transit" for IPv6. We see almost identical distributions for the port 0 and other ports.

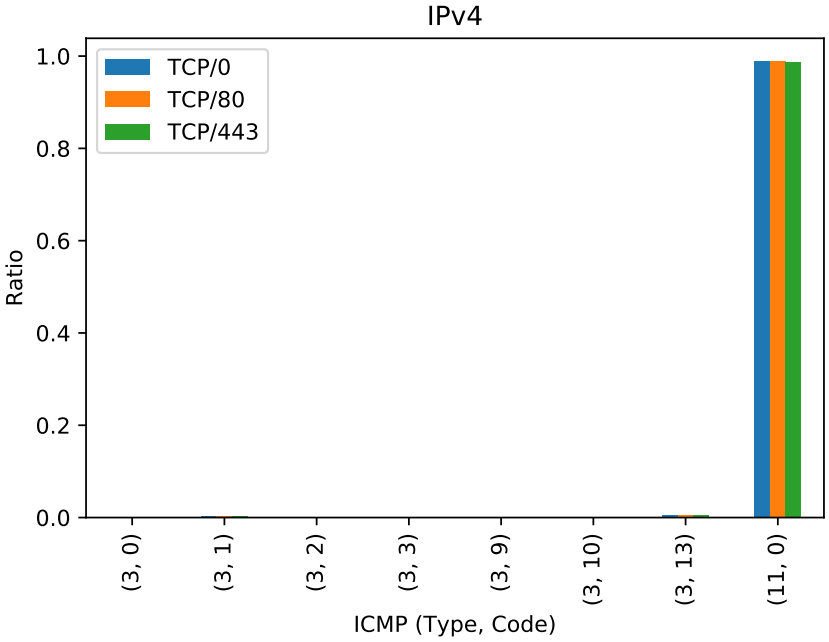


(a) Last responsive hop

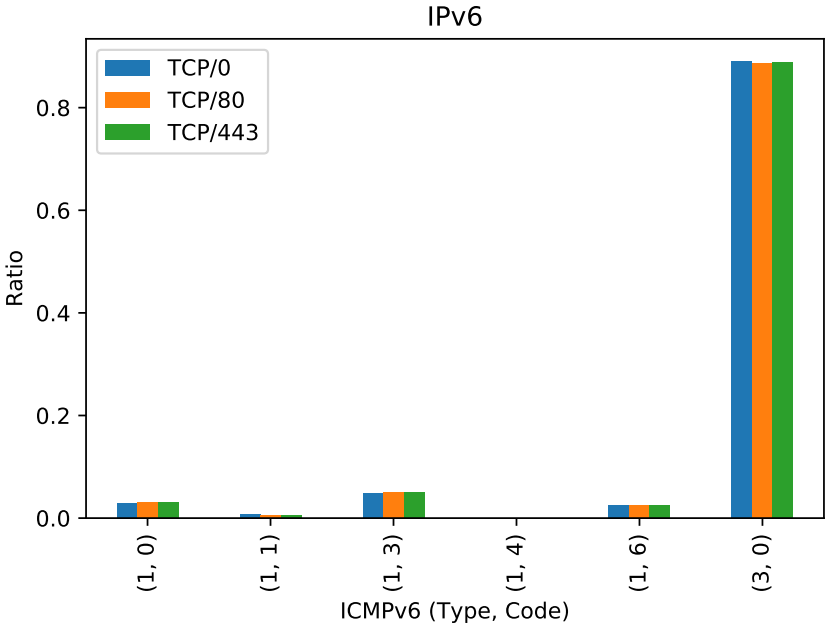


(b) Number of responsive hops

Figure 5.9: Box plot of (a) last responsive hop, and (b) number of responsive hops (right) aggregated by transport port protocol for IPv4 and IPv6 showing the median, first and third quantiles, mean (▲), and 1.5 times IQR as whiskers



(a) IPv4



(b) IPv6

Figure 5.10: Distribution of ICMP(v6) type and code combinations for all responses split by transport protocol

5.6 Summary

In this chapter, we dissected port 0 traffic by analyzing five complementing passive datasets and by conducting active measurements. We showed that the majority of port 0 traffic in the wild flows between a small number of source and destination ASes/IP addresses. Moreover, for some ASes we identified similar diurnal patterns in port 0 traffic as with regular traffic, along with many TCP packets with no TCP flags, hinting at a prevalence of fragmented traffic in the IXP dataset. Additionally, we found that a major fraction of UDP port 0 traffic contains payload, with BitTorrent being a common contributor. Moreover, we showed that TCP port 0 traffic usually does not contain any payload and is mostly one-way. Two-way streams were identified as mostly scanning traffic. Finally, by staging an active measurement campaign, we showed unusually high response rates to TCP port 0 probes in IPv4, in addition to uncovering the presence of port 0 packet filtering.

In previous chapters, we investigated Internet traffic monitoring. Throughout this Chapter, we investigated a minor traffic category in the Internet, namely, the traffic using port 0 and characterized it. In the next chapter, we broaden our knowledge regarding a more prevalent form of traffic category, namely the encrypted traffic using VPN protocols. We try to find the VPN servers in the wild, and then characterize the Internet traffic going to them in Chapter 6.

6

Characterizing the VPN Ecosystem in the Wild

With the increase of remote working during and after the COVID-19 pandemic, the use of Virtual Private Networks (VPNs) around the world has nearly doubled. Therefore, measuring the traffic and security aspects of the VPN ecosystem is more important now than ever. VPN users rely on the security of VPN solutions, to protect private and corporate communication. Thus a good understanding of the security state of VPN servers is crucial. Moreover, properly detecting and characterizing VPN traffic remains challenging, since some VPN protocols use the same port number as web traffic and port-based traffic classification will not help.

In Chapter 6, we aim at detecting and characterizing VPN servers in the wild, which facilitates detecting the VPN traffic. To this end, we perform Internet-wide active measurements to find VPN servers in the wild, and analyze their cryptographic certificates, vulnerabilities, locations, and fingerprints. We find 9.8M VPN servers distributed around the world using OpenVPN, SSTP, PPTP, and IPsec, and analyze their vulnerability. We find SSTP to be the most vulnerable protocol with more than 90% of detected servers being vulnerable to TLS downgrade attacks. Out of all the servers that respond to our VPN probes, 2% also respond to HTTP probes and therefore are classified as Web servers. Finally, we use our list of VPN servers to identify VPN traffic in a large European ISP and observe that 2.6% of all traffic is related to these VPN servers.

Virtual Private Networks (VPNs) provide secure communication mechanisms, including encryption and tunneling, enabling users to circumvent censorship, to access geo-blocked services, or to securely access an organization's resources remotely.

The COVID-19 pandemic changed Internet traffic dramatically. Studies investigating the impact of the COVID-19 pandemic on Internet traffic show that streaming traffic being tripled around the world due to remote work, remote learning, and entertainment services [185–187]. VPN traffic has been no exception to this major traffic shift. After the COVID-19 pandemic, the VPN traffic observed in a large European IXP nearly doubled [13]. In a campus network, even a more dramatic increase of 20x has been reported [187], which shows a prominent growth of remote work and e-learning. Additionally, several articles find that remote work is here to stay [188, 189]. According to recent statistics from SurfShark [190], 31% of all Internet users use VPNs.

In order to facilitate network planning and traffic engineering, ISPs have an interest in understanding the network applications being used by their clients, and how these applications behave in terms of traffic patterns and volume. Therefore, detecting and characterizing VPN traffic is an important task for ISPs. Certain VPN protocols use known port numbers for their operation, e.g., port number 4500 is used for IPsec, and port number 1723 is used for SSTP. Thus, the traffic using protocols over the known port numbers can easily be detected as VPN traffic. However, some VPN protocols, e.g., SSTP, and in some occasions, OpenVPN use port number 443 which is commonly used for secure web applications. This makes it challenging to distinguish between web and VPN traffic.

Moreover, VPN users might share sensitive private or corporate data over VPN connections. As the number of cyber attacks has almost doubled after the pandemic [191], it makes Internet users even more aware of their privacy and the security of their VPN connections. Therefore, investigating the vulnerabilities of the VPN protocols helps to highlight existing shortcomings in VPN security.

Previous studies focused on detecting VPN traffic using machine learning [192, 193], or DNS-based approaches [13, 194]. Some studies have also analyzed the commercial VPN ecosystem [195, 196]. However, to the best of our knowledge, this is the first work which conducts active measurements to detect and characterize VPN servers in the wild.

In this chapter, we aim to detect, characterize, and analyze the deployment of VPN servers in the Internet using active measurements along with passive VPN traffic analysis. We also make use of the system proposed in Chapter 4, namely, FlowDNS to collect more information about the VPN traffic.

Specifically, this work makes the following main contributions:

- **VPN server deployment:** We perform active measurements to the complete IPv4 address space and an IPv6 hitlist for 4 different VPN protocols both in UDP and TCP. We find around 9.8 million IPv4 addresses and 2.2 thousand IPv6 addresses responsive to our probes.
- **VPN security evaluation:** We analyze the detected IP addresses in terms of TLS vulnerabilities, certificates, and geolocation. We observe that the United States is the most common location among our detected IP addresses. We also find that more than 90% of SSTP servers are vulnerable to a TLS attack and nearly 7% of the certificates are expired.
- **VPN traffic analysis:** We analyze passive traffic traces from a large European ISP, we find that 2.6% of the traffic uses our list of VPN servers as either source or destination address. Moreover, we use rDNS data along with DNS records from a large European ISP to compare our results with previous work looking into VPN classification [13]. We find that using our methodology, we find 4 times more VPN servers in the wild.
- **VPN probing tool:** We develop new modules for Zgrab [21] to send customized VPN probes. We make these modules publicly available [197] to foster further research in the VPN ecosystem.

6.1 Background

VPNs establish cryptographically secured tunnels between different networks and can be used to connect private networks over the public network. Thus, a proper VPN connection should be encrypted in order to prevent eavesdropping and tampering of VPN traffic. The tunneling mechanism of a VPN connection also provides privacy since the traffic is encapsulated. Therefore, users remotely accessing a private network appear to be directly connected.

While the exact tunneling process varies depending on the underlying VPN protocol, it is quite common to categorize VPNs in two different groups:

- **Site-to-site VPNs**: In this configuration, a VPN is used to connect two or more networks of geographically distinct sites. This is common for companies with branches in different locations.
- **Remote access VPNs**: This kind of VPN connection is mainly used by individual end-users in order to connect to a private network.

6.1.1 VPN Usage

The usage of VPNs has evolved over the past three decades. David Crawshaw [198] gives a very comprehensive overview of how and why VPNs changed over the years. While in the earlier days of the Internet, they were primarily used by companies to connect their geographically distinct offices, VPNs nowadays provide a variety of use cases for individuals as well and are used by millions of end-users around the globe. Use cases include:

- **Privacy preservation**: The encrypted VPN tunnels provide end-users the means to preserve their privacy.
- **Censorship circumvention/accessing geo-blocked content**: Specific services might be censored in some countries or geographically restricted. By connecting to a VPN server in a different country, it is still possible to access such content since it would now appear as if the user was located in a different country.
- **Remote access**: It is common to use VPNs to remotely access restricted resources or to connect with an organization's network. This usage scenario has gained importance especially during the COVID-19 pandemic among employees and students alike due to remote working.

Different usage patterns, the general understanding of the functionality of VPNs, and awareness of potential risks vary between different demographic groups. Dutkowska-Zuk et al. [199] studied how and why people from different demographic backgrounds use VPN software primarily comparing the general population with students. They found that the general population is more likely to rely on free, commercial VPN solutions to protect their privacy. Students, on the other hand, rather resort to VPN software for remote access or to circumvent censorship and access geographically blocked services with an increased use of institutional VPNs. Generally, they found

VPN protocol	Transport protocol	Port	(D)TLS-based	Server detection possible
IPsec/L2TP	UDP	500	✗	✓
OpenVPN	UDP & TCP	1194, 443	✓	partially
SSTP	TCP	443	✓	✓
PPTP	TCP	1723	✗	✓
AnyConnect	UDP & TCP	443	✓	✗
WireGuard	UDP	51820	✗	✗

Table 6.1: Overview of VPN protocols showing the transport protocol, port, (D)TLS encryption, and possible detection

that, while most VPN users are concerned about their privacy, they are less concerned about data collection by VPN companies.

Especially during the COVID-19 pandemic, VPNs increasingly gained significance. The pandemic and the resulting lockdowns caused many employees and students to work and study remotely from home. Feldmann et al. [13] analyzed the effect of the lockdowns on the Internet traffic. Their work included the analysis of how VPN traffic shifted during the pandemic. They detected a traffic increase of over 200% for VPN servers identified based on their domain with increased traffic even after the first lockdowns. These findings highlight the rising significance of VPNs. With progressing digitalization, VPN traffic can be expected to increase even further.

6.1.2 VPN Protocols

We want to cover as many protocols as possible including some of the most prominent ones like OpenVPN and IPsec. The functionality of a VPN connection establishment varies depending on the underlying VPN protocol. Table 6.1 gives an overview of all the VPN protocols we consider with general information on their underlying protocols. Among them, especially PPTP, which was the first actual VPN protocol standardized in 1999 (see RFC 2637 [200]), can be considered rather outdated and it is not recommended to be used anymore [198, 201].

WireGuard is the most modern protocol at the moment. It is much more simplistic than, e.g., OpenVPN or IPsec and incorporates state-of-the-art cryptographic principles.

6.2 Methodology

In this section, we introduce our methodology for our passive and active measurements. We perform Internet-wide measurements in order to detect VPN servers in the wild and create hit lists of identified VPN servers. Based on those results, we conduct follow-up measurements to fingerprint the VPN servers and further analyze them in terms of security. Finally, we look for the detected IP addresses in the traffic from a large European ISP to find out the amount of VPN traffic.

6.2.1 VPN Server Detection

Our measurements to detect VPN servers include the whole IPv4 address range as well as over 530 million non-aliased IPv6 addresses from the *IPv6 Hitlist Service* [23, 202]. We send out the connection initiation requests that are used in the connection establishments of the different VPN protocols. For UDP-based protocols, we use ZMap [19] (ZMapv6 for IPv6 [20]), a transport-layer network scanner, to directly send out UDP probes. If the VPN protocol is TCP-based, we first use ZMap to find targets with the respective open TCP ports using TCP SYN-scans. We then use Zgrab [21] to send out the actual VPN requests. Zgrab works on the application layer. It can be used complementary to ZMap for more involved scans. It also allows us to implement custom modules needed for our VPN requests over TCP and TLS.

We identify an address as a VPN server based on the responses we receive to our initiation requests. If the parsed response satisfies the format of the expected VPN response, the target is classified as a VPN server. To completely detect the VPN ecosystem for a specific server, we might have to take several server configurations into account and perform multiple measurements for a single protocol accordingly. Apart from that, for some protocols and configurations, we require knowledge of cryptographic key material which we do not have since we perform measurements in the wild. Therefore, we cannot detect the entirety of the VPN ecosystem with our method. The last column of Table 6.1 summarizes for which protocol we are able to detect VPN servers. When OpenVPN servers specify the so-called *tls-auth* directive, an HMAC signature is required in all control messages. This means that we can only craft requests without HMACs and hence detect only a subset of all OpenVPN servers.

As mentioned above, for some protocols, it might also be necessary to consider different configurations. For IPsec, e.g., we suggest seven different cipher suites in the initiation request. Apart from that, we have to specify a key exchange method in the OpenVPN requests. Out of the two possible key exchange methods, namely *key method 1* and *key method 2*, key method 1 is considered insecure and is therefore deprecated [203]. We therefore specify key method 2 in our initiation requests and then perform a follow-up scan where we suggest the deprecated key exchange method to identified OpenVPN servers to investigate how many of them might still support key method 1.

6.2.2 TLS Analysis

For the TLS-based VPN protocols, which include SSTP and OpenVPN over TCP, we perform follow-up measurements to further fingerprint the servers and assess them in terms of security. For that, we collect TLS certificates of the VPN servers to analyze them for expiry, check for self-signed certificates and investigate how many of them are snake oil certificates. We characterize a certificate as a snake oil certificate if the common name (CN) of the subject and issuer are both specified as *localhost* or *user.local*. For the certificates signed by a Certificate Authority (CA), we collect the

most common issuing organizations. We gather domain names corresponding to the responsive IP addresses using reverse DNS (rDNS) look-ups, and collect certificates with and without the Server Name Indication (SNI) extension using these domain names and compare them against each other. SNI can be used by the client in the TLS handshake in order to specify a hostname for which a connection should be established. This might be necessary in cases where multiple domain names are hosted on a single address. Finally, we test if the servers are susceptible to the *Heartbleed* [204] vulnerability as well as a series of TLS downgrade attacks. The Heartbleed attack is based on the Heartbeat Extension [205] of the OpenSSL library. In TLS downgrade attacks, we try to force a server to establish a connection using an outdated SSL/TLS version or using insecure cipher suites by suggesting those outdated primitives in the TLS handshake. Table 6.6 summarizes all the vulnerabilities and their requirements, i.e., what we have to test for or the version or cipher suite to which we try to downgrade the TLS connection. For instance, in order to check if a server is vulnerable to the FREAK attack, we suggest any SSL/TLS version and only RSA_EXPORT cipher suites in the TLS handshake.

6.2.3 Fingerprinting

We try to infer more information on the VPN servers based on our connection initiation requests as well as from follow-up measurements in order to further categorize them.

One aspect we examine is the server software deployment. For SSTP and PPTP, we can extract information on the software vendor directly from the responses to our initiation requests.

Furthermore, we perform OS detection measurements on a subset of 1000 VPN servers for each protocol using *Nmap* [206], a network scanner that can be used for network discovery among other things. We use Nmap's *fast* option and target 100 instead of 1000 ports to decrease runtime and parse the results for the most common open ports and OS guesses. With those results, we can learn more about the VPN server infrastructure and potential other services running on the same servers.

6.2.4 VPN Traffic Analysis

The active measurement in Section 6.2.1 provides us with a list of IP addresses, namely VPN hitlist, which are responsive to at least one VPN protocol initiation request. We look for these IP addresses in the DNS records gathered from the DNS resolvers at a large European ISP during a 1-hour period to learn about the domain names these IP addresses are associated with. We do not expect to find all the detected IP addresses in these DNS responses. Therefore, for any remaining IP address, we use reverse DNS resolution to find the corresponding domain names.

Then, we look for the IP addresses from our VPN hitlist on over a week of network flow data from the ISP to find out the amount of traffic associated with the VPN

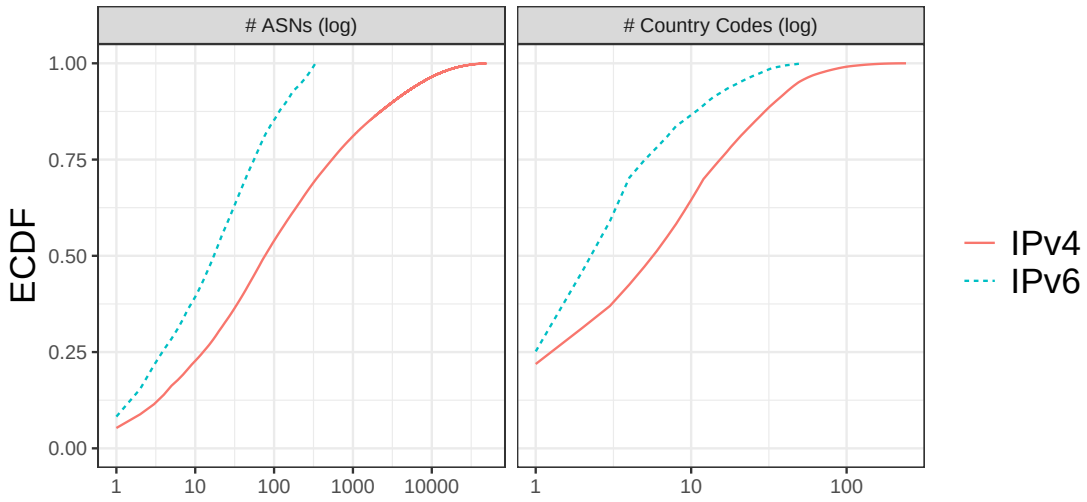


Figure 6.1: Cumulative distribution of number of ASes (left) and number of countries (right) corresponding to the responsive IPs

hitlist and compare the results with a port-based VPN traffic detection, and also a state-of-the-art approach.

6.2.5 Ethical Considerations

As explained in Section 2.3, we best current practices both for active and passive measurements. We plan to notify the VPN providers about their servers' vulnerabilities.

6.3 Active Measurements of the VPN Server Ecosystem

In this section, we go through the results from our Internet-wide active measurement using different VPN protocols. We discuss the characteristics of the responsive servers such as geographical locations, VPN protocols, etc. Then, we analyze their vulnerabilities and try to fingerprint them based on the gathered information.

6.3.1 Responsive Servers

In total, we find 9,817,450 responsive IPv4 addresses with our probes that we can identify as VPN servers.

rDNS. We investigate the reverse DNS records corresponding to the responsive IPv4 addresses. We aggregate results on the second-level domain and sort them based on the number of responsive IPs that they correspond to. We find that all the top 10

(a) IPv4

AS number	AS name	VPN servers
4134	ChinaNet	515,830
7922	Comcast	356,327
1221	Telstra	257,821
3320	Deutsche Telekom	242,433
4766	Korea Telecom	228,863
4713	NTT Communications	145,286
7018	AT&T	137,698
4837	China Unicom	133,861
3462	HiNet	119,612
20115	Charter Communications	97,109

(b) IPv6

AS number	AS name	VPN servers
7922	Comcast	183
63949	Akamai	159
12322	Proxad Free SAS	138
7506	GMO Internet Group	89
9009	M247 Ltd	63
9370	Sakura Internet Inc	58
14061	DigitalOcean	55
2516	KDDI Corporation	54
7684	Sakura Internet Inc	39
680	DFN-Verein	36

Table 6.2: AS numbers, AS names and number of VPN servers belonging to the ASes for IPv4 and IPv6

domain names belong to telecommunication companies (e.g., Open Computer Network, a large Japanese ISP, and Telstra, an Australian telecommunications company). Next, we filter all rDNS records which contain *vpn* in their second-level domain names in order to detect commercial VPN providers. We find a single domain related to PacketHub which manages IP addresses for several companies, including NordVPN, a major commercial VPN provider. This domain name ranks 60th among all rDNS second level domains.

AS analysis. Figure 6.1 shows the distribution of ASes to which our responsive IP addresses belong. The responsive IP addresses are originated by 49625 and 334 ASes in total, while top 10 ASes contribute to 22% and 38% of the IP addresses, for IPv4 and IPv6 respectively, as shown in Figure 6.1. Top 10 ASes for IPv4 responsive addresses are all telecommunication companies, while out of the top 10 ASes for IPv6 responsive addresses, 8 are telecommunication companies and 2 are academy-related ASes. Tables 6.1(a) and 6.1(b) further summarize the top 10 AS numbers as well as the AS names or organizations and the number of VPN servers that are registered within the respective AS. As can be seen, most top ASes are large ISP networks.

Moreover, we investigate the top ASes for commercial VPN providers. As shown by Ramesh et al. [196] it is quite common for commercial VPN providers to use shared infrastructure. 27 providers, including popular companies such as NordVPN, Norton Secure VPN, or Mozilla VPN, use the same AS, namely AS 9009 operated by M247 Ltd. This AS is also visible in our measurements and it ranks 14th with 74,894 identified VPN servers (0.76% of all addresses). Furthermore, Ramesh et al. [196] find that some IP blocks in AS 16509 (Amazon) are shared across Norton Secure VPN and SurfEasy VPN. AS 16509 lands on rank 20 of our list being shared by almost 60,000 VPN servers (0.6% of all addresses). Another AS known to be used by VPN providers is AS 60068—again operated by M247 Ltd.— which is used by NordVPN and CyberGhost VPN. It ranks on place 178 of our list with 6,898 VPN servers (0.07% of all addresses). Overall, we find that although the top ASes are dominated by large ISPs, a considerable number of VPN servers are located in ASes used by commercial VPN providers.

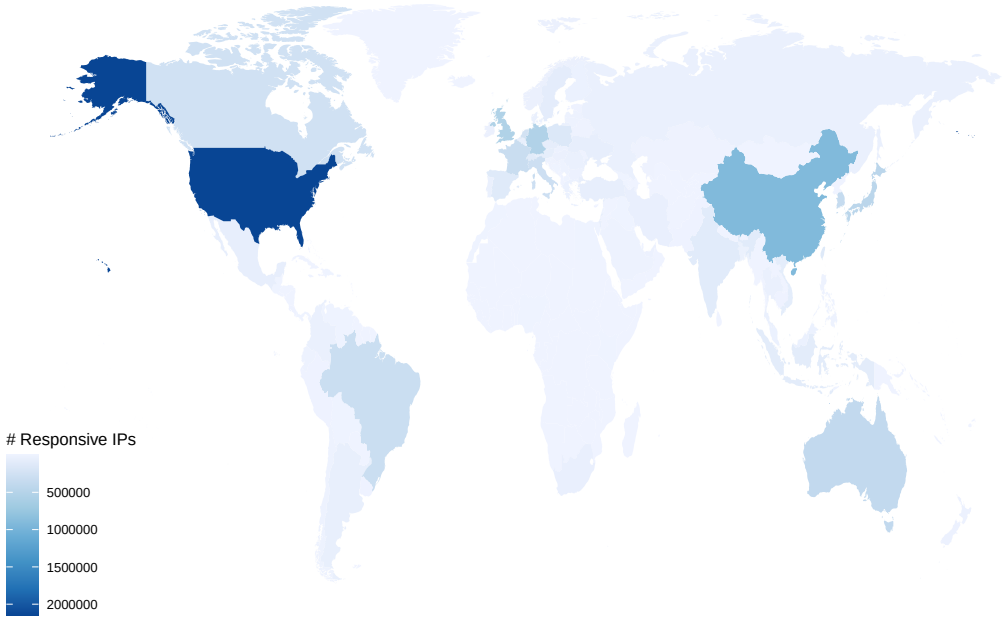


Figure 6.2: Geographical distribution of responsive IPv4 addresses per country

VPN protocol	Detected servers
IPsec	7,008,298
PPTP	2,424,317
OpenVPN	1,436,667
SSTP	187,214

Table 6.3: Number of detected VPN servers per protocol

Geolocation. We use Geolite Country Database [207] to determine the location of the responsive IP addresses. Figure 6.2 shows a heatmap of the number of responsive IPv4 addresses per country. We observe that responsive IP addresses are scattered all over the world, in total over 241 and 52 countries for IPv4 and IPv6, respectively. However, 64% and 86% of IP addresses belong to the top 10 countries for IPv4 and IPv6 respectively. Top 3 countries contributing to IPv4 responsive addresses are the United States, China, and UK, while top 3 countries for IPv6 are the United States, Japan, and Germany.

6.3.2 VPN Protocols

We are able to detect servers for IPsec, PPTP, OpenVPN without tls-auth, and SSTP. Table 6.3 summarizes our findings. Our IPsec UDP probes yield by far the most responsive VPN servers. It might seem surprising that we find such a large number of PPTP servers in contrast to OpenVPN and SSTP considering that PPTP

is far more outdated and OpenVPN is one of the most prominent VPN protocols. However, we have to keep in mind that we can only detect a subset of the whole OpenVPN ecosystem since some configurations require knowledge of cryptographic key material as explained in Section 6.2.1. Apart from that, SSTP can only be used for remote access connections, whereas PPTP used to be the most widely deployed VPN protocol. We can assume that a large number of the detected PPTP servers are quite outdated, yet still running.

Out of the around 1.4 million OpenVPN servers, 1,011,178 were detected over UDP and 482,956 over TCP. Considering that the TCP version of OpenVPN is generally rather considered as a fallback option, this disparity is to be expected. Figure 6.3 visualizes the intersection of those two address sets in a Venn diagram. We can see that the majority of the servers supports only a single transport protocol.

Overlap between protocols. In the next step, we compare the IP address sets for the four protocols to depict their intersections and to find out how many of the servers support more than one VPN protocol. Figure 6.4 summarizes those findings in an upset plot. The horizontal bars on the left visualize the sizes of the four protocol sets. The vertical bars represent the different intersections and the sets to be considered are indicated by the black dots below the vertical bars. The first bar on the left, e.g., represents the number of VPN servers supporting both PPTP and IPsec with roughly 550,000 servers making up for around 5.7% of the whole detected VPN server ecosystem. The second bar on the right, on the other hand, represents the number of servers supporting all four protocols, which is close to zero with only around 2.8 thousand servers.

We can see that the majority of all VPN servers support only one of the four protocols we consider in this work. Since commercial VPN providers usually offer a variety of different VPN protocols to choose from, it is possible that a large percentage of the servers supporting several protocols are commercial. This might be the case especially for the ones supporting three or four protocols. We investigate the rDNS records corresponding to the servers supporting all the four protocols, and find that there are no commercial VPN provider in the top 10 second-level domains. All in all, we find that commercial VPN providers account for only a fraction of the entire VPN server ecosystem considering the supported protocols.

Different protocol versions. Some VPN protocols might include different versions or configurations, like OpenVPN, for instance. We therefore try to trigger VPN responses from OpenVPN servers suggesting the outdated key exchange method key method 1. We also try to trigger responses with random HMAC signatures.

We find that only 84 of the roughly 1.4 million servers accept our random signature. Apart from that, none of the detected servers support the insecure key exchange method. While most of the servers ignored our requests, we still received around 6,500 responses specifying the default key exchange method key method 2. We can therefore conclude that key method 1 is truly deprecated in the OpenVPN ecosystem.

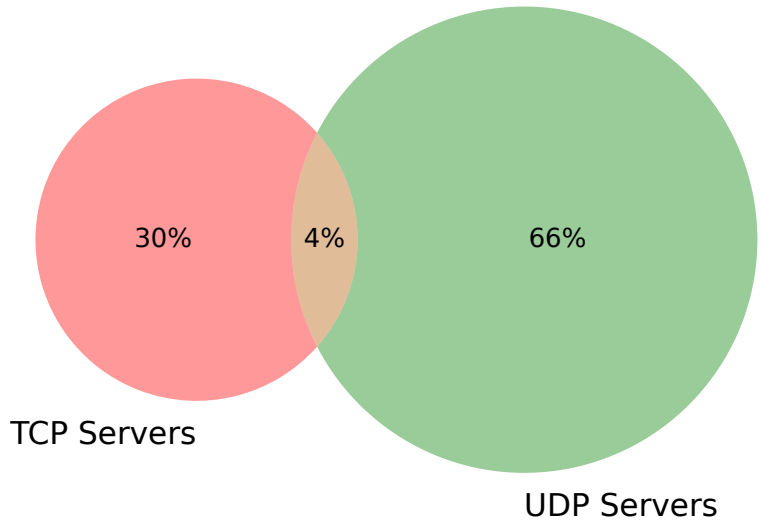


Figure 6.3: Intersection of OpenVPN UDP and TCP servers

6.3.3 Security Analysis

TLS certificate analysis. We collect TLS certificates for the TLS-based VPN servers which include SSTP and OpenVPN over TCP and consider only unique certificates. For that, we compare the certificate fingerprints, i.e., the unique identifier of the certificate, to make sure we do not consider the same certificate more than once. Some certificates, however, do not include a fingerprint. Therefore, the number of certificates that we analyze in the end might be higher than the number of unique certificates. For OpenVPN, we find 129,143 unique certificates with a fingerprint for 312,095 servers. The most frequently occurring certificate is collected over 10,000 times and is issued for *www.update.microsoft.com*. For SSTP, there are 104,988 fingerprints for 184,047 servers. We detect a certificate issued for **.vpnauction.com* 2561 times and one for **.trust.zone* 1194 times. These are commercial VPN providers that seem to use the same certificate for all of their VPN servers. While we are able to collect certificates for nearly all of the SSTP servers, we only receive TLS certificates for around 65% of the detected OpenVPN servers. This is most likely caused by the fact that OpenVPN performs a variation of the standard TLS handshake during connection establishment. Therefore, some of the servers might not respond when trying to initiate a regular TLS handshake.

Table 6.4 summarizes the results of the certificate analysis and contains the number of certificates that we analyzed after filtering out unique certificates and certificates without fingerprints. We detect a large number of self-issued or self-signed certificates for both protocols. Out of the self-issued certificates, we characterize only around 4.7% as snake oil certificates for SSTP and close to zero for OpenVPN with around 0.4%. However, 33% of the self-issued SSTP certificates contain *softether*, an open-source and multi-protocol VPN software, in the CN fields. 13% specify an IPv4 address in the CN sections. Upon looking at the organization field, we find over

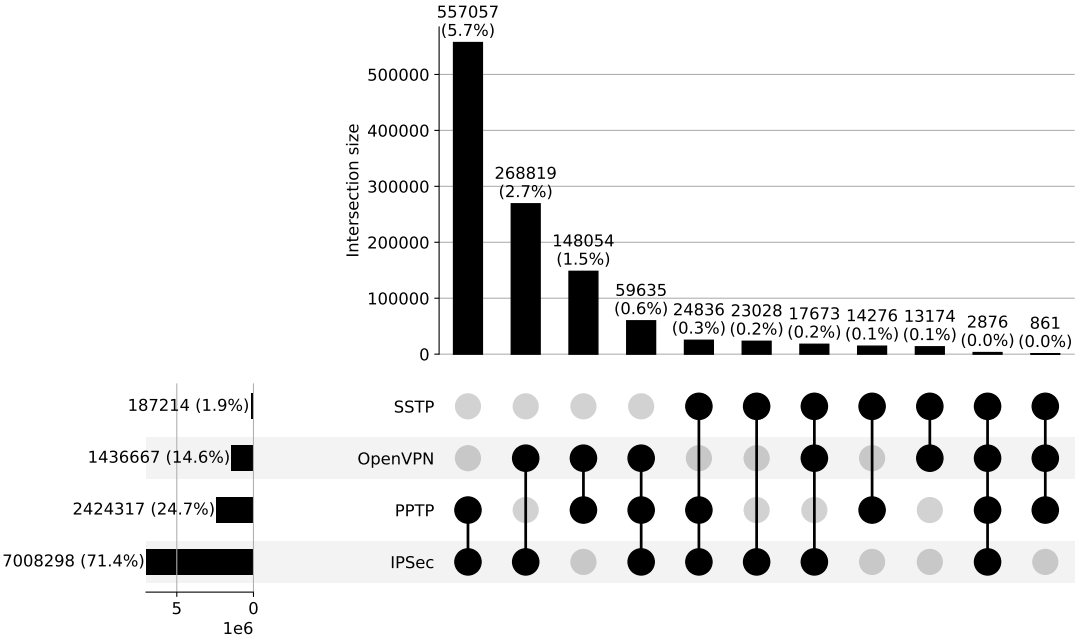


Figure 6.4: VPN protocol summary: Number of detected VPN servers for each protocol and the intersection between all protocols

	OpenVPN TCP	SSTP
Expired	6080 (3.8%)	13,370 (9%)
Self-issued	109,965 (69%)	39,889 (28%)
Self-signed	109,825 (69%)	34,725 (24%)
All certificates	158,705	143,517

Table 6.4: Expired, self-issued, and self-signed TLS certificates for OpenVPN and SSTP

21,000 different organizations where almost 14,000 specify no organization at all. For the OpenVPN certificates, we find that around 77% of the self-issued certificates include the *Fireware web CA* as CNs specifying *WatchGuard* as organization. For the rest, we detect more than 21,000 different organizations.

Looking at the organization fields of the CA-signed certificates, we can learn more about the signing authorities. Considering SSTP, we filter out 2502 different organizations for almost 100,000 CA-signed certificates. Table 6.4(b) contains the top five organizations accounting for 87% of all signings. We examine the issuer CNs for the certificates that do not specify an organization, yet we could not find any meaningful information with 7,531 different issuers and the most frequently occurring CN being *CA* with 159 signings. For OpenVPN, the organizations are a lot more heterogeneous with 14,548 organizations in total. The top five organizations in Table 6.4(a) account for only around 50% of all signings.

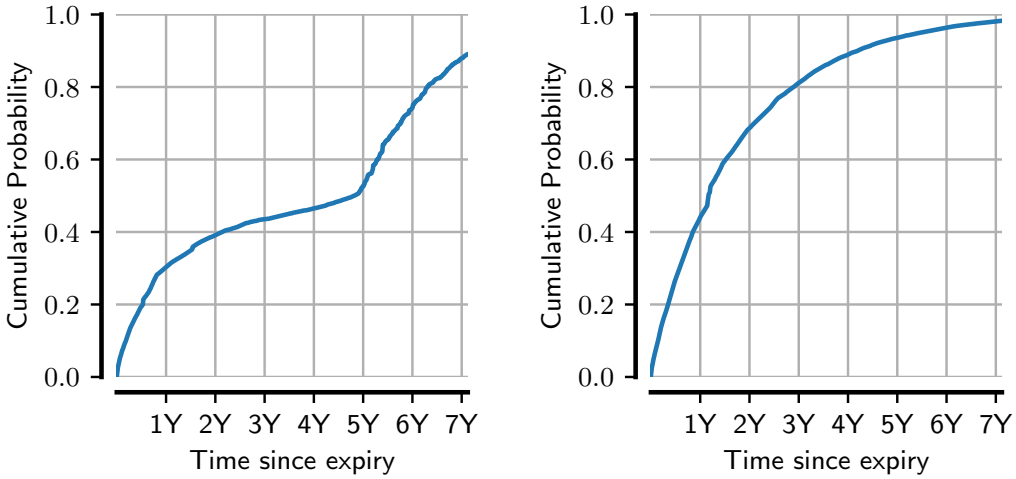
(a) OpenVPN

Issuer	Certificates
Stormshield	8966
Let's Encrypt	7463
Sectigo Limited	2610
Digicert Inc.	1897
GoDaddy.com, Inc.	1332

(b) SSTP

Issuer	Certificates
DigiCert, Inc.	45,156
Sectigo Limited	12,795
GoDaddy.com, Inc.	10,958
N/S	9139
Let's Encrypt	7801

Table 6.5: Certificate issuer distribution for OpenVPN and SSTP servers



(a) OpenVPN server certificates.

(b) SSTP server certificates.

Figure 6.5: Distribution of expiry time (time between day of expiry and Aug. 15, 2022) for expired certificates

	TLS version	Cipher suites	Other requirements	OpenVPN	SSTP
RC4 [208]	All	RC4	None	32,294	84,892
Heartbleed [204]	All	All	OpenSSL Heartbeat	232	10
Poodle [209]	SSL 3.0	All	None	7,005	24,917
FREAK [210]	All	RSA_EXPORT	None	31	1
Logjam [211]	All	DHE/512-bit export	None	8	0
DROWN [212]	SSLv2	All	None	0	0
ROBOT [213]	All	TLS_RSA	None	95,301	174,986
Raccoon [214]	TLS $\leq$ 1.2	TLS_DH	None	0	0

Table 6.6: Requirements for TLS vulnerabilities and number of vulnerable servers per protocol

Since we also detect a quite significant number of expired certificates, we examine the date of their expiry more thoroughly. Figure 6.5 shows ECDFs for the time that has passed since the dates of expiry and the 15th of August, 2022. In general, over half of the SSTP certificates expired over a year ago. For OpenVPN it is even around 70%. It is possible that those certificates belong to outdated, forgotten VPN servers.

TLS vulnerability analysis. The results of our TLS vulnerability analysis for the TLS-based VPN protocols can be found in the last two columns of Table 6.6 where we count the occurrences of susceptible servers. We detect a larger number of vulnerable servers for RC4, Poodle and ROBOT for both protocols, yet only a few outliers for the rest. SSTP is much more likely to show signs of vulnerability for all three attacks with over 90% of the servers being susceptible to ROBOT. This is most likely caused by the fact that SSTP is based on an outdated version of SSL and highlights why SSTP is not recommended to be used anymore.

The effect of not using SNI. As we target only IP addresses in our follow-up TLS measurements without the SNI extension, we want to investigate the effect of not using SNI. Therefore, we first perform an rDNS resolution for our IP addresses and find 259,910 domain names for about 480,000 OpenVPN TCP servers and 86,630 domain names for roughly 180,000 SSTP servers. We now collect certificates with the SNI extension and then re-run the TLS scans without SNI for the respective addresses for whose domains we could gather certificates.

Table 6.7 shows the results of the comparison of those two types of certificates and the number of certificates we could collect. Two certificates mismatch when the fingerprints differ. We then compare different fields and summarize the mismatch occurrences in the table. If those fields match and the certificate has only been renewed, we do not count it as a mismatch.

While the results for both protocols are similar, relatively speaking, we find more mismatches for SSTP. About 3% mismatch for OpenVPN, whereas for SSTP 5.5% mismatch. To confirm that those mismatches are caused by using SNI in the TLS handshakes, we perform a second measurement without SNI and compare the certificates with the other non-SNI results. Without SNI, we find less than half as many mismatches for SSTP and more than three times fewer mismatches for OpenVPN.

	OpenVPN	SSTP
SNI Certificates	84,212	45,405
no SNI Certificates	81,379	45,026
Certificate Mismatches	2491	2515
Authority Key ID Mismatches	2051	1463
Subject Key ID Mismatches	2407	2379
Subject SANs Mismatches	2008	1677
Issuer CN Mismatches	1933	1476
Subject CN Mismatches	2021	1627

Table 6.7: Comparison of Certificates Collected with and without SNI

Considering the overall number of certificates from our large-scale measurements compared to the ones we collected with SNI and keeping in mind the mismatches we detected in the two non-SNI measurements, we can conclude that not using SNI affects less than 1% of the certificates for both protocols and the effect is therefore negligible.

6.3.4 Fingerprinting

Server Software. For SSTP and PPTP, we can infer the server-side software from the responses we receive to our initiation requests. For SSTP, we find that around 80% of all detected servers use *Microsoft HTTPAPI 2.0*. Around 19% use *MikroTik-SSTP* and less than 1% use something else or specify nothing at all.

However, the PPTP vendor software is a lot more heterogeneous compared to SSTP. Table 6.8 shows the different software vendors we detect in the VPN server responses. While there are four prominent vendors, over 15% of the PPTP servers rely on 183 different types. This can have potential security implications on the PPTP ecosystem. Assuming there was some kind of new vulnerability, the rollout of a security update to counter this vulnerability would be significantly slower compared to SSTP with fewer software vendors. A similar phenomenon where vendor fragmentation leads to slower update rollout can also be observed in the Android ecosystem. Thomas et al. [215] showed that almost 60% of all devices ran insecure Android versions in July 2015. This share declines only slowly after the discovery of a major vulnerability. They found out that the bottleneck of this issue lies with the manufacturers and results in 87.7% of all devices being exposed to at least 11 critical vulnerabilities. Jones et al. [216] considered manufacturers between 2015 and 2019 and further showed that the median latency of a security update is 24 days with an additional latency of 11 days before an end-user update.

Nmap OS detection and port scans. In our Nmap OS detection measurements, we first have a look at the most common ports for all four protocols. Figure 6.6 summarizes the most frequently occurring open ports. As expected, the default HTTP(S) ports 443 and 80 are the most common ports, with the exception of the PPTP servers

Vendor	Percentage
Linux	32.3%
MikroTik	30.6%
Draytek	21.1%
Microsoft	6.9%
Cananian	2.0%
Fortinet PPTP	1.4%
Yamaha Corporation	1.4%
Cisco Systems, Inc.	1.2%
Others (162)	3.2%

Table 6.8: Software vendors for detected PPTP servers

for which the default PPTP port TCP/1723 obviously is the most widely used port. As Ramesh et al. [196] pointed out, specific open ports do not pose security risks by themselves, yet, they might still be abused in order to identify and exploit particular services [217].

For the OS detection, we filter out the first guesses for every target and look at the most common OSes and version ranges:

- **IPsec:** We receive 48 unique first guesses for 126 hosts out of 722 responsive IPsec servers. Out of those, 40 guess the Linux Kernel ranging from version 2.6.32-3.10. In general, Linux is the most common OS with 67 guesses. However, Microsoft was barely guessed as an OS vendor with only nine guesses.
- **PPTP:** For 792 responsive hosts, Nmap was able to guess an OS for 216 addresses with 56 unique guesses. Linux was once again the primary occurrence. Out of those guesses, 88 specified Linux 2.6.32-3.10, where the majority mewlie below version 3.2, however. As for IPsec, we have very few results for Microsoft with only 15 guesses. For the PPTP servers, there were more hardware guesses compared to the other protocols with 36 guesses specifying some kind of hardware device.
- **OpenVPN:** The most frequent guesses are almost exclusively Linux again in 33 unique guesses for 89 out of the 763 responsive hosts. 39 specify Linux ranging from 3.2-4.11, i.e., the versions are not quite as outdated as for PPTP and IPsec. We received only a single guess for Microsoft products.
- **SSTP:** The SSTP scans result in 44 different guesses for 178 out of 948 responsive hosts. This time, we have more results for Microsoft products with a total of 49 guesses. The most prominent vendor is Linux again, however, with 101 guesses where 53 range from Linux versions 2.6.32-3.10.

6.3.5 IPv6

VPN server detection. Targeting roughly 530 million IPv6 addresses in our ZMapv6 port scans, we could detect 1,195,510 responsive hosts on port TCP/443

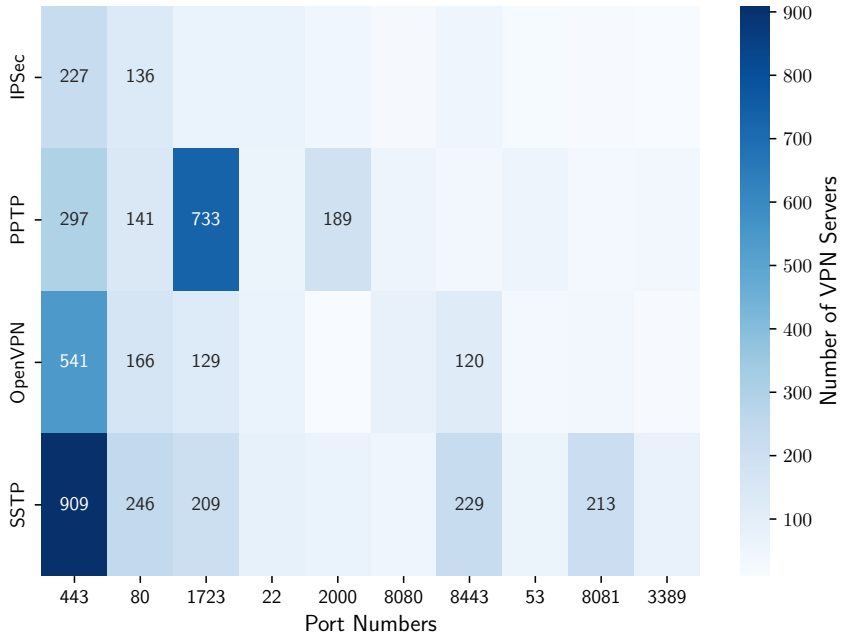


Figure 6.6: Heatmap of most frequently detected open ports per VPN server

which we target in our follow-up Zgrab scans for SSTP and OpenVPN over TCP. We could not find any responsive addresses on port TCP/1723, the default PPTP port. Since port TCP/1723 is used exclusively for PPTP and the protocol is very outdated, it is not too surprising that there are no IPv6 servers supporting PPTP. Apart from that, we do not get any responses on the UDP ports 500 (IPsec) and 1194 (OpenVPN over UDP).

Out of the roughly 1.2 million hits on port TCP/443, we could identify 2070 addresses as OpenVPN servers and 949 as SSTP servers with a total of 2221 VPN servers supporting IPv6. While those results seem very low, we have to keep in mind that the rollout of IPv6 is still very slow in general. IPv6 is also not yet supported by most commercial VPN providers.

As also observed in IPv4 results in Section 6.3.2, none of the OpenVPN servers accepted our OpenVPN key method 1 requests with only 11 servers still responding with the secure key exchange method. Additionally, of the overall IPv6 VPN servers we detect, around 36% support both protocols, i.e., compared to IPv4, the overlap is higher.

Investigating the rDNS records corresponding to the responsive IPv6 addresses, we observe that the top 10 domains belong to hosting providers, cloud providers, and research networks. Similar to the IPv4 results, we do not find a domain name belonging to a commercial VPN provider among the top 10 domains. By filtering second-level domains to match **vpn** we find the commercial VPN provider WhiteLabel VPN, ranking 25th among the top domains.

Therefore, we infer that most of the VPN servers that support IPv6 are, in fact, not commercial VPN providers.

TLS certificate analysis. The results of the TLS certificate analysis are similar to IPv4. We could collect certificates for around 75% of the identified OpenVPN servers with 816 unique fingerprints. Combined with the certificates that do not contain a fingerprint, we analyze a total of 1882 certificates. We collected certificates for every SSTP server resulting in 747 certificates after filtering out 207 unique fingerprints. Less certificates are expired this time with only 3.3% for OpenVPN and 2.1% for SSTP. This time, only 29% of the OpenVPN certificates are self-signed. For SSTP, more certificates are self-signed for the IPv6 servers with over 70% of all certificates. Out of those, we characterize roughly 2% as snake oil certificates for both protocols. Furthermore, about two thirds of the self-signed certificates for both protocols were issued by softether.

When examining the signing organizations for the CA-signed certificates, we find that around 85% (709 certificates) of the OpenVPN certificates are signed by Let's Encrypt with a total of 43 organizations. For SSTP, around 73% are signed by Let's Encrypt (153 certificates). Here, we find a total of only 16 organizations.

TLS vulnerability analysis. The results of the TLS vulnerability analysis are very similar to the IPv4 VPN servers. For both protocols, we are only able to detect vulnerable servers for the same three prominent attacks as for the IPv4 analysis. Out of the 2070 OpenVPN servers, 31% are vulnerable to RC4 biases, 6% to Poodle and 74% to Robot. When analyzing the 949 SSTP servers, we find that 67% are vulnerable to RC4 biases, 13% to the Poodle attack and roughly 98% to ROBOT. While the results are similar to our large-scale measurements, we can conclude that the VPN servers supporting IPv6 are much more likely to show any signs of vulnerability with the vast majority being vulnerable to the ROBOT attack.

The effect of not using SNI. The rDNS measurements for the IPv6 servers resulted in 410 domain names for SSTP and 813 domain names for OpenVPN over TCP. Again, we first collect TLS certificates using the SNI extension and then try the same without SNI and compare the results. We find that only around 3% of the certificates for both protocols mismatch in terms of fingerprints and important certificate fields including authority and subject key IDs, subject SANs, and CNs. When comparing those results by running a second TLS scan without SNI, we find that only around 2.5% of the OpenVPN and less than 1% of the SSTP certificates differ. Considering the overall number of certificates, the effect of not using SNI is even less significant compared to IPv4 and is therefore negligible.

VPN server software. Since we could not analyze the PPTP server software ecosystem this time, we can only compare the results for SSTP. The results are similar again with 91% of the SSTP servers specifying the Microsoft HTTP API 2.0. However, the rest did not specify any vendor, i.e., the IPv6 SSTP servers seem to not use MikroTik-SSTP with Microsoft being the only vendor.

Nmap OS detection and port scans. As for IPv4, we perform Nmap measurements on the detected IPv6 VPN servers including 1000 random OpenVPN TCP

servers and all 949 SSTP servers. Out of those servers, 874 OpenVPN servers and 852 SSTP servers are responsive.

The most commonly used open port is TCP/443 with 838 occurrences (96%) for OpenVPN and 852 (97%) for SSTP. Compared to IPv4, the number of open HTTPS ports is much higher for OpenVPN. Here, we have to keep in mind that we can only consider OpenVPN servers over TCP. Thus, this disparity is to be expected. The second most frequently open port for both protocols, in contrast to IPv4, is TCP/22, the default SSH port. This port occurs 245 times (28%) for OpenVPN and even 391 times (46%) for SSTP. Other common ports for both protocols are ports TCP/8000 for OpenVPN (21%) and TCP/80 accounting for around 17% of the open ports for both protocols.

We receive more OS guesses for the IPv6 servers compared to IPv4. As was the case for IPv4, we filter out the first guesses for every target:

- **OpenVPN**: The measurement results in only four unique guesses for a total of 481 hosts. 93% specify Linux with 416 guessing Linux 3.X and 33 guessing version 2.6. Only 19 predictions include a Microsoft OS and only 13 an Apple product.
- **SSTP**: For SSTP, there are five unique predictions for 406 addresses. The majority specifies Linux again with 91%. Out of those, 333 guesses specify Linux version 3.X and only 36 specify version 2.6. Microsoft OSes are predicted 36 times and only a single guess specifies a macOS.

In contrast to IPv4, Nmap was able to predict an OS for a much larger percentage of our targets with an OS guess for almost half of the targets. Additionally, the predictions are a lot more homogeneous. Linux is again the most prominent vendor, however, the predicted versions are not quite as outdated as for the IPv4 servers.

6.4 Passive VPN Traffic Analysis

It is important for network operators and ISPs to gain insight over the volume and daily patterns of VPN traffic. In a previous study, Feldmann et al. [13] try to find the VPN traffic based on the domain names corresponding to the IP addresses observed in the traffic. For detecting VPN traffic, Feldmann et al. use domain names to infer whether the IP addresses corresponding to them carry VPN traffic. They exclude any domain name that starts with *www.*, and does not have **vpn** to the left of the public suffix. Finally, they consider the remaining domain names as VPN domain names and count the traffic that relate to these domain names as VPN traffic.

To compare our methodology with the state of the art, we apply the methodology used by Feldmann et al. [13] on our results. We use DNS responses gathered by DNS resolvers at a large European ISP, and look for those DNS responses that include the IP addresses from our VPN hitlist. We find 13% of the IP addresses from the VPN hitlist in the above-mentioned DNS responses. Therefore, we complement our DNS data with reverse DNS look-ups for all the remaining IP addresses. To refine

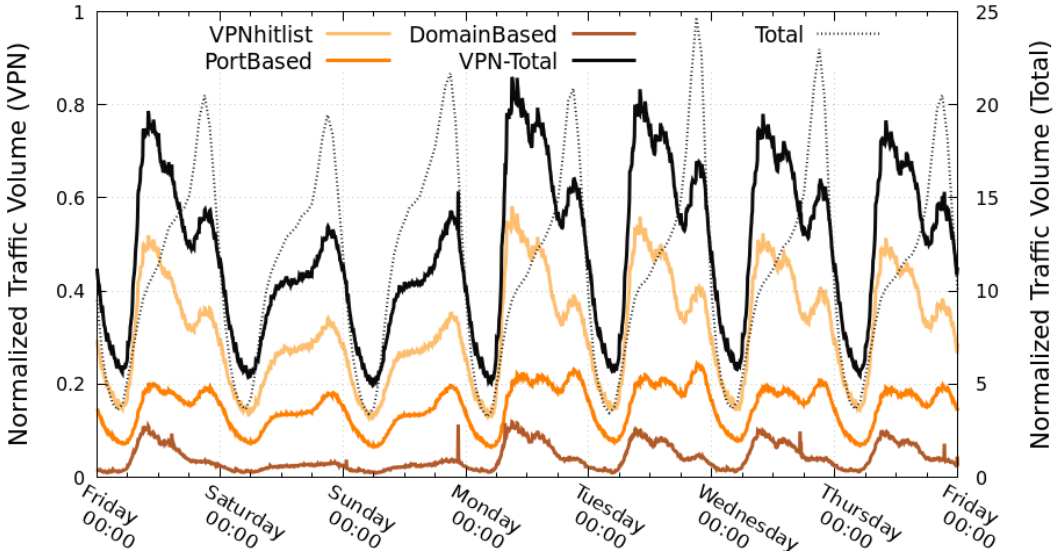


Figure 6.7: Normalized VPN traffic volume for different traffic detection techniques

the reverse DNS results, we exclude any domain names containing any order of the corresponding IP address bytes or octets in decimal or hexadecimal format. Overall, we end up with the domain names corresponding to 23.6% of the IP addresses from the VPN hitlist. Then, we apply the methodology used by Feldmann et al. on the resulting domain names, i.e., we extract those domain names that contain **vpn** on the left side of the public suffix [218], while excluding any domain starting with www. to exclude web servers. We observe that this methodology captures only 4.8% of our VPN hitlist. Therefore, our approach can detect 4 times more VPN servers compared to the methodology by Feldmann et al.

Finally, we look at a one-week snapshot of all the network flow traffic from the large European ISP to find out the amount of traffic that can be attributed to VPN.

To this end, we compare the amount of VPN traffic detected with three methodologies:

1. *VPN Hitlist*: the methodology proposed in this disseration, i.e., sending active probes, including the responsive IP addresses in a hitlist, excluding those IP addresses that answer to web requests, i.e., HTTP GET requests, then measuring the traffic volume originated by or destined to these IP addresses.
2. *Port-based*: this methodology captures the traffic only based on port numbers, considering traffic with port numbers 500 (IPsec), 4500 (IPsec), 1194 (OpenVPN), 1701 (L2TP), 1723 (1723) both on UDP and TCP as VPN traffic.
3. *Domain-based*: the methodology proposed by Feldmann et al, i.e., filtering domain names based on certain keywords, then measuring the traffic volume originated or destined to the IP addresses corresponding to these domain names.

Figure 6.7 shows the traffic volume considered as VPN traffic by each of the above-mentioned methodologies. The solid black line shows the total amount of VPN traffic detected by either of the three approaches. The dashed line shows the total traffic volume in the ISP. The left Y axis shows the VPN traffic volume (including all the three approaches), and the right Y axis shows the total ISP traffic volume. All the traffic values are normalized. While normalizing, we keep the ratio between the VPN traffic and total traffic intact. Therefore, comparing the left and right axis values shows that the total traffic is roughly 25 times as much as all VPN traffic.

Compared to the *Port-based* approach, we detect twice as much traffic, and compared to the *Domain-based* approach, we detect 8 times as much using the *VPN Hitlist*.

The mean VPN traffic volume detected by all three approaches is 4.1% of the mean total ISP traffic over the week, with *VPN Hitlist* contributing to 2.6%, *Port-based* 1.3%, and *Domain-based* 0.3%.

Looking at the overlap between every two approaches, we find that only 2.7% of all the traffic detected by all three approaches is detected both by *VPN Hitlist* and *Domain-based*. We observe 1.2% overlap between the traffic detected by *VPN Hitlist* and *Port-based* approaches.

We observe a diurnal pattern in the VPN traffic detected by all of the three approaches. We find that VPN traffic pattern in the weekdays differs from that of weekends. It peaks at noon in the weekdays, and at night in the weekend, while the total ISP traffic always follows the same pattern, i.e., peaks at night. It could indicate the fact that the VPN traffic is mostly work-related through weekdays, while mostly entertainment-related throughout the weekend. In the domain-based approach the amount of VPN traffic detected by the *Domain-based* approach is much less in the weekends than in the weekdays. This could indicate that the *Domain-based* approach detect mostly work-related VPN servers.

We investigate the domain names corresponding to the traffic we detect using our approach and find that *vpn.*, *mail.*, *www.*, and *remote.* are among the most common prefixes left to the public suffix part of the domain names, with *vpn.* being the most common prefix. The fact that we observe *mail.* and *www.* might be either re-use of the same domain name for other purposes by the network operators, or a mislabeling effect from our approach caused by not answering our HTTP Get requests. Also, looking at the DNS records corresponding to the IP addresses from our hitlist, using FlowDNS— the system to correlate DNS and Netflow data at scale [2] which we discussed in Chapter 4—we find that 5 out of 10 top domains are related to commercial VPN providers and the rest are CDN domains. We observe that the most common source port/destination port combination is 4500/4500 which belongs to IPsec, also port number 1194 which is registered for OpenVPN, and at the same time 1193, which is practically used for VPN [219]. We also observe that 51820/51820 and 1337/1337 which belongs to WireGuard protocol are among the top port number pairs observed in the traffic detected by our approach. Port 51820 also falls into the range of ephemeral ports numbers (49152 to 65535) which can be temporarily used by many applications. However, due to the prominent existence of this port number

in our results accompanied with port 1337, we infer that we can possibly detect some WireGuard traffic, although in our active measurement approach we cannot scan the WireGuard protocol. This might be due to the co-existence of multiple protocols in one VPN server. This shows that although in our approach we cannot scan WireGuard protocol, we can still detect some WireGuard traffic which might be due to the co-existence of multiple protocols on one VPN server. Traffic related to WireGuard protocol contributes to 8.6% of the detected VPN traffic by our VPN hitlist, while contributing to only 2% of the traffic detected by the *Domain-based* approach.

6.5 Discussion

In this work, we detect VPN servers in the wild by sending Internet-wide active probes using different VPN protocols. We can distinguish between VPN servers and Web servers by excluding those servers that respond to a Web request. We compare the amount of traffic detected by our approach and two other approaches over a week of traffic from a large European ISP and find out that the approach proposed by this work detects much more VPN servers compared to the state-of-the-art domain-based approach. In addition, our approach benefits from detecting VPN servers that do not use any domain name, and can also detect VPN traffic that is using unusual ports in case these servers answer VPN probe on the usual VPN port numbers. Also, to be the best of our knowledge, this is the first work to perform an Internet-wide active measurement of VPN servers in the wild.

VPN hitlist. We send active probes according to the specification of VPN protocols including SSTP, PPTP, OpenVPN, and IPsec to the whole IPv4 address space and to an IPv6 hitlist. We make our list of detected VPN servers, namely the VPN hitlist, publicly available at vpnecosystem.github.io. This VPN hitlist can be useful for network operators to find out about the amount and patterns of VPN traffic in their networks. The VPN hitlist can also be used by fellow researchers to investigate different behaviors of the VPN servers and VPN traffic, e.g., investigating actual attacks to these servers.

Security. We also investigate the security of the OpenVPN and SSTP protocols in terms of different security aspects, including heartbleed attack, TLS certificates security, and TLS downgrade attacks. We find that SSTP servers use expired certificates 3x more than OpenVPN servers. We also find that 90% of the SSTP servers are vulnerable to ROBOT attack. Therefore, we find SSTP to be the most vulnerable protocol. This striking high percentage of vulnerable servers for some of the protocols shows, that the VPN server ecosystem is not as secure as some users believe it to be. Therefore, we hope that our analysis can highlight these security risks with using each VPN protocol and also helps network operators choose the right VPN protocols for their networks.

Limitations. Our approach builds upon receiving answers from the servers in the wild and therefore, has its limitations. If there is a VPN protocol which uses

a pre-shared key in the first VPN request and does not respond otherwise, we are unable to detect it. Examples of such VPN protocols are WireGuard and Cisco AnyConnect. Therefore, we are unable to detect any VPN server which offers only these two protocols. However, we observe that 8.6% of the detected traffic is related to WireGuard which might be due to multiple protocols being served by one VPN server. In addition, certain VPN servers might only work on non-registered port numbers for better anonymization. Since in our work, we only send probes to the port numbers registered for the VPN protocols by IANA [220], we cannot detect VPN servers that work on unusual port numbers. Therefore, our list of detected VPN servers is limited to those using the supported VPN protocols and working on their registered port numbers.

Future work. In the future, our work can be complemented by including more port numbers in the active scans. Results from previous studies on predicting the services across all ports [221] can be used together with our approach to gain more coverage. Despite the above-mentioned limitations, our proposed approach detects much more VPN servers compared to the state-of-the-art domain-based approach, and also, to the best of our knowledge, is the first work to perform an Internet-wide active measurement of VPN servers in the wild.

Reproducibility. We make our analysis code and data [222], customized Zgrab modules [197], and our VPN hitlist publicly available[1] for fellow researchers to be able to reproduce our work and build upon it.

6.6 Related Work

VPN traffic classification is an open research problem, particularly challenging due to its encrypted nature. There are several studies trying to tackle this problem using machine learning approaches. Some are able to categorize the traffic into VPN and non-VPN only [223], and some provide more detailed sub-categories [224–226]. Zou et al. [224] identify encrypted traffic by combining a deep neural network to extract features of single packets and a recurrent network to analyze features of the traffic flow based on features of three consecutive packets. Though the model classifies some traffic incorrectly regarding sub-categories, it could achieve almost 99% accuracy when only considering VPN and non-VPN traffic. Alfayoumi et al. [226], on the other hand, also consider time-related features and subdivide traffic by also identifying applications.

All of these works require previously captured unencrypted VPN traffic to train. Previous studies have also tried to detect VPN traffic using the DNS records corresponding to the IP addresses observed in the traffic [13].

In this chapter, we propose a different approach, i.e., Internet-wide active measurements, to detect VPN servers in the Internet. Internet-wide measurements have been previously applied for several intents including finding IPv6 responsive addresses [23],

[1] `https://vpnecosystem.github.io/`

responsive IPs to abnormal traffic [227], the usage of DNS over encryption [228], and so on. However, to the best of our knowledge, this is the first work applying active measurements to detect VPN servers in the wild and detecting the traffic based on a VPN hitlist.

Investigating the security of the VPN servers is also an interesting research problem which is already addressed by several studies. For example, Xue et al. investigate the possibility and practicality of fingerprinting OpenVPN flows [229]. Tolley et al. investigate the vulnerability of known VPN servers to spoofed traffic [230]. Crawshaw [198] addresses vulnerabilities that come with some of the protocols themselves, such as outdated cryptographic cipher suites used in PPTP. In his proposal for WireGuard [231], Donenfeld talks about disadvantages in current popular VPN protocols. *VPNalyzer* requires a tool to be installed on the user's device to measure and collect data on the active VPN connections in terms different security aspects including data leakage, open ports, and DNSSEC validation [196]. Appelbaum et al. also identified vulnerabilities of commercial and public online VPN servers [232].

A large body of literature also exists that empirically examines TLS vulnerabilities including self-signed root CA injection to intercept TLS connection [233, 234], and improper implementation of the protocol making version downgrade attacks possible even with new TLS 1.3 [235].

We mainly focus on potential vulnerabilities that come with VPN protocols which are built on top of SSL/TLS. Thus, we investigate SSL/TLS related features of those protocols. For some identified OpenVPN servers, we can also make assumptions on their security based on information we can infer about their server configurations. All the previous works study the security of known VPN servers, while in this dissertation, we measure the vulnerability of our detected VPN server in the Internet.

6.7 Summary

In this chapter, we performed the first Internet-wide active measurement on the VPN server ecosystem for OpenVPN, SSTP, PPTP, and IPsec both in IPv4 and IPv6 to detect VPN servers in the wild. We detected 9.8 million VPN servers distributed globally. 10% of the detected VPN servers offered more than one VPN protocol with very few serving all the four protocols we studied. We also send active Web probes to the detected VPN servers and observed that 2% were both VPN and Web servers. Analyzing the TLS-based VPN protocols, i.e., OpenVPN and SSTP, we found that SSTP was the most vulnerable to a version downgrade attack, and certificates of OpenVPN servers had the most self-signed and self-issued certificates. We also tried to fingerprint the detected VPN servers in terms of server software vendors and operating systems. Finally, using our VPN hitlist, excluding the servers that were both VPN and Web servers, we observed that VPN traffic constitutes 2.6% of the total traffic volume in a large European ISP, which is 8x as much as that of a state-of-the-art domain-based approach, and twice as much as the trivial port-based approach. We publish our VPN hitlist, our customized ZGrab2 modules for VPN

scans, and the code to our analysis for future researchers and network operators to use.

7 Conclusion

Monitoring Internet traffic is important for the network operators to gain insights about the traffic going through their networks for troubleshooting and optimizing their networks. In the meanwhile, it helps Internet users to know more about the security of the data they exchange on the Internet.

Monitoring Internet traffic characteristics such as volume, intent, and pattern is non-trivial due to encryption, large volume, and time-criticality. There are also some important traffic patterns, e.g., VPN traffic or port 0 traffic, that are less studied. The overarching goal of this thesis, namely, *uncovering hidden characteristics of Internet traffic*, refers to using novel methodologies to study the underlying or less apparent aspects of Internet traffic.

Throughout this thesis, we strived to facilitate Internet traffic monitoring by designing novel systems to perform large-scale network monitoring in real-time. These systems handle a substantial amount of network flow data and enable real-time analysis of network flow characteristics. This information when put together with our Internet-wide active measurement methodologies, help us gain a comprehensive view of different characteristics of Internet traffic.

7.1 Summary

As explained in Chapter 2, ISPs and IXPs usually capture sample Internet flows for operational purposes. For conducting the research upon which this dissertation is founded, we collaborated with a large Tier-1 European ISP and a large European IXP to process those sample flows and evaluate our systems on their data and premises.

Exploring Network-wide Flow Capture Data. In Chapter 3, we strived to answer our first research question: *How can we monitor the traffic using the existing network flow captures in near real-time with a priori unknown queries?* To answer this question, we built a system called Flowyager to store, process, and query existing flow traffic captures. Using Flowyager, we generated and analyzed tree data structures called Flowtrees that were succinct summaries of the 5-tuple raw flow data available by capture utilities. Using these self-adjusted data structures, we observed a drastic reduction in space and transfer requirements, by 75% to 95%, compared to raw flow records. Flowyager manages the storage and transfers of Flowtrees, supports Flowtree operators, and provides a structured query language, called FlowQL

for answering flow queries across sites and time periods in near real-time. We then deployed a Flowyager prototype at a large Internet Exchange Point and a Tier-1 Internet Service Provider to showcase its capabilities for networks with hundreds of router interfaces. We showed that the query response time can be reduced by an order of magnitude when compared with alternative data analytics platforms, e.g., ClickHouse and Spark. We demonstrated that with the help of all the above-mentioned modules, Flowyager enables *interactive network-wide queries* and offers unprecedented drill-down capabilities to, e.g., identify DDoS culprits, pinpoint the involved sites, and determine the length of the attack.

FlowDNS: Correlating Netflow and DNS Streams at Scale. Knowing flow characteristics through processing the 5-tuple helps identify DDoS attacks. However, knowing the purpose or the content of the traffic flows, and more specifically the service they are originated by, helps network operators to better plan their networks to enhance the performance exactly where the customer's interests lie, and also offer the customers relevant commercial packages. However, with the increasing deployment of CDNs by different services, identification, and attribution of the traffic on network-layer information alone becomes a challenge: If multiple services are using the same CDN provider, they cannot be easily distinguished based on IP prefixes alone. Therefore, it is crucial to go beyond pure network-layer information for traffic attribution. For this reason, in Chapter 4, we tried to answer our second research question: *How can we recognize the application (domain name) a certain traffic flow is using?* To answer this question, we leveraged real-time DNS responses gathered by the clients' default DNS resolvers. We designed a system to correlate these DNS responses with network-layer headers to translate CDN-hosted domains to the actual services to which the traffic belongs. We deployed this system at a large European ISP and observed that we could correlate an average of 81.7% of the traffic with the corresponding services, without any loss on our live data streams. Our correlation results also showed that 0.5% of the daily traffic contained malformatted, spamming, or phishing domain names. Moreover, to showcase the usage of such results for ISPs, we correlated the results with BGP information to find more details about the origin and destination of the traffic. We also published our correlation software for other researchers or network operators to use.

Zeroing in on Port 0 Traffic in the Wild. Using the system built in Chapter 3, we found out that there is a large amount of abnormal traffic. Internet services leverage transport protocol port numbers to specify the source and destination application layer protocols. While using port 0 is not allowed in most transport protocols, we observed a non-negligible share of traffic using port 0 in the Internet. Therefore, Chapter 5 studies the third research question: *How can we characterize port 0 traffic? Why do we see port 0 traffic?* We studied port 0 traffic to understand the origins and causes of such traffic to answer this question. Since Flowyager alone was not enough to grasp all the different aspects of such traffic, we used five complementing flow-level and packet-level datasets. We observed 73 GB of port 0 traffic in one week of IXP traffic, and we identified most of them to be an artifact of packet fragmentation. In our packet-level datasets, most traffic is originated from a small number of hosts and while most of the packets have no payload, a major fraction of packets containing

payload belong to the BitTorrent protocol. Moreover, we found unique traffic patterns commonly seen in scanning. In addition to analyzing passive traces, we also conducted an active measurement campaign to study how different networks react to port 0 traffic. We found an unexpectedly high response rate for TCP port 0 probes in IPv4, with very low response rates with other protocol types. We provided the results to the measurement community.

Characterizing the VPN Ecosystem. With the increase of remote working during and after the COVID-19 pandemic, the use of Virtual Private Networks (VPNs) around the world has nearly doubled. Therefore, measuring the traffic and security aspects of the VPN ecosystem is more important now than ever. VPN users rely on the security of VPN solutions, to protect private and corporate communication. Thus a good understanding of the security state of VPN servers is crucial. Moreover, detecting and characterizing VPN traffic remains challenging, since some VPN protocols use the same port number as web traffic and port-based traffic classification will not help.

FlowDNS helped us analyze different sources of Internet traffic to infer the service or domain name to which the traffic belonged. However, relying only on the domain name to infer whether VPN is used is not enough.

Therefore, the research question motivating the work in Chapter Chapter 6 was the following: *How can we characterize the encrypted VPN traffic?* To answer this, we aimed at detecting and characterizing VPN servers in the wild to facilitate detecting the VPN traffic. To this end, we performed Internet-wide active measurements to find VPN servers in the wild, and analyze their cryptographic certificates, vulnerabilities, locations, and fingerprints. We found 9.8M VPN servers distributed around the world using OpenVPN, SSTP, PPTP, and IPsec, and analyze their vulnerability. We observed that SSTP was the most vulnerable protocol with more than 90% of the detected servers being vulnerable to TLS downgrade attacks. Out of all the servers that responded to our VPN probes, 2% also responded to HTTP probes and therefore were classified as Web servers. Finally, we used our list of VPN servers to identify VPN traffic in a large European ISP and observed that 2.6% of all traffic was related to these VPN servers.

7.2 Future Work

In this section, we present the future directions for our work. We address the future directions corresponding to each work in their separate paragraph. By addressing these future directions, we aim to deepen our understanding of network traffic patterns, and enhance network security.

Exploring Network-wide Flow Capture Data. We proposed a system to monitor network-wide flow data in Chapter 3 and evaluated its accuracy and performance with specific parameters. Future work can explore the accuracy and performance of Flowyager by using different space restrictions and maintaining more counters in

Flowtrees. Moreover, although Flowyager is designed to be used in distributed mode, operating on different network nodes, it has not been evaluated in such a setup which is a possible future work direction. Finally, combining the results from Flowyager with other sources of data, e.g., DNS data to dig deeper into the desired flows can uncover more characteristics of Internet traffic.

FlowDNS: Correlating Netflow and DNS Streams at Scale. We introduced FlowDNS in Chapter 4 and showcased some of its applications. This system can be used to analyze the change in the behavior of Internet traffic from/to a specific website when important events occur. In future, further research can be carried out to analyze the behavior of Twitter traffic before and after Elon Must took over Twitter, or to analyze the behavior of DisneyPlus traffic during an important live stream. This is especially useful for websites that do not use their own CDNs and therefore, their traffic is not recognizable without knowing the domain name to which they belong. Using FlowDNS, further research can be conducted to cross-check the traffic patterns with the peering agreements to assess the amount to which peering agreements are followed in the real world.

Zeroing in on Port 0 Traffic in the Wild. In Chapter 5, we investigated port 0 traffic in the wild and the origins of such traffic. For future research projects, specifically, those using NetFlow or IPFIX flow data, we highlight the need to be aware of port 0 flows that are an artifact of packet fragmentation. Also, it is crucial for network administrators to be aware of port 0 traffic and avoid it to mitigate the chance of getting attacks on this port number. In this work, we found out that BitTorrent is a prominent source or destination of port 0 traffic. Future studies can combine the flow or packet captures with other sources of data such as DNS data to investigate if specific domain names contribute to a high fraction of port 0 traffic either as source or destination.

Characterizing the VPN Ecosystem. We introduced our approach to characterize the VPN ecosystem in Chapter 6. Since VPN protocols are meant to provide anonymity, they may use non-famous port numbers for their operation. In the future, our work can be complemented by including more port numbers in the active scans. Results from previous studies on predicting the services across all ports [221] can be used together with our approach to gain more coverage.

By utilizing the methodologies and systems proposed in this dissertation, future researchers can delve further into Internet traffic characteristics. For example, they can explore application-specific traffic analysis, traffic volume growth, and uncover patterns of malicious traffic, such as DDoS attacks and malware propagation.

Throughout this thesis, we conducted studies on streaming Internet traffic and VPN traffic. However, there is still a wide range of Internet traffic characteristics that need to be uncovered. Different types of Internet traffic exhibit distinct characteristics and requirements. These include Internet of Things (IoT) traffic, mobile internet traffic, satellite Internet traffic, online transactions, and social media traffic. For instance, online transactions necessitate specific reliability and security considerations that need to be addressed. Future research can focus on assessing the extent to which

these considerations are met. From an Internet traffic perspective, there is also a need to examine the traffic patterns and encryption types utilized by these applications.

Bibliography

[1] Aniss Maghsoudlou, Oliver Gasser, and Anja Feldmann. "Zeroing in on Port 0 Traffic in the Wild". In: *Passive and Active Measurement*. Ed. by Oliver Hohlfeld, Andra Lutu, and Dave Levin. Cham: Springer International Publishing, 2021, pp. 547–563. ISBN: 978-3-030-72582-2 (cit. on pp. ix, 5).

[2] Aniss Maghsoudlou, Oliver Gasser, Ingmar Poese, and Anja Feldmann. "FlowDNS: Correlating Netflow and DNS Streams at Scale". In: *Proceedings of the 18th International Conference on Emerging Networking EXperiments and Technologies*. CoNEXT '22. Roma, Italy: Association for Computing Machinery, 2022, pp. 187–195. ISBN: 9781450395083 (cit. on pp. ix, 5, 54, 103).

[3] Aniss Maghsoudlou, Lukas Vermeulen, Ingmar Poese, and Oliver Gasser. "Characterizing the VPN Ecosystem in the Wild". In: *Passive and Active Measurement*. Ed. by Anna Brunstrom, Marcel Flores, and Marco Fiore. Cham: Springer Nature Switzerland, 2023, pp. 18–45. ISBN: 978-3-031-28486-1 (cit. on pp. ix, 5).

[4] Said Jawad Saidi, Aniss Maghsoudlou, Damien Foucard, Georgios Smaragdakis, Ingmar Poese, and Anja Feldmann. "Exploring Network-Wide Flow Data With Flowyager". In: *IEEE Transactions on Network and Service Management* 17.4 (2020), pp. 1988–2006 (cit. on pp. ix, 5).

[5] Aniss Maghsoudlou, Oliver Gasser, and Anja Feldmann. *Reserved: Dissecting Internet Traffic on Port 0*. Poster at Passive and Active Measurement Conference 2020. 2020 (cit. on pp. ix, 5, 65, 66).

[6] *Global digital population as of April 2022*. `https://www.statista.com/statistics/617136/digital-population-worldwide/`. July 2022 (cit. on p. 1).

[7] *Gone in Minutes, Out for Hours: Outage Shakes Facebook*. `https://www.nytimes.com/2021/10/04/technology/facebook-down.html`. Oct. 2021 (cit. on p. 1).

[8] *How Verizon and a BGP Optimizer caused a major internet outage affecting Amazon, Facebook, CloudFlare among others*. `https://hub.packtpub.com/how-verizon-and-a-bgp-optimizer-caused-a-major-internet-outage-affecting-amazon-facebook-cloudflare-among-others/`. June 2019 (cit. on p. 1).

[9] Elias Bou-Harb, Mourad Debbabi, and Chadi Assi. "On Fingerprinting Probing Activities". In: *Computers & Security* 43 (2014), pp. 35–48. ISSN: 0167-4048 (cit. on pp. 2, 66).

[10] M. Luchs and C. Doerr. "The Curious Case of Port 0". In: *Proceedings of the IFIP Networking Conference*. 2019, pp. 1–9 (cit. on pp. 2, 65, 66).

[11] IANA. *Service Name and Transport Protocol Port Number Registry*. 2020. URL: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml (cit. on pp. 2, 65).

[12] Aumnat Tongkaw. "VPN Security in Campus Network during Covid-19 epidemic: Case Study in Southeast Asia". In: *2021 International Conference on Electrical, Computer and Energy Technologies (ICECET)*. 2021, pp. 1–6 (cit. on p. 8).

[13] Anja Feldmann, Oliver Gasser, Franziska Lichtblau, Enric Pujol, Ingmar Poese, Christoph Dietzel, Daniel Wagner, Matthias Wichtlhuber, Juan Tapiador, Narseo Vallina-Rodriguez, Oliver Hohlfeld, and Georgios Smaragdakis. "The Lockdown Effect: Implications of the COVID-19 Pandemic on Internet Traffic". In: *Proceedings of the ACM Internet Measurement Conference*. IMC '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1–18. ISBN: 9781450381383 (cit. on pp. 8, 83, 84, 86, 101, 105).

[14] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. "Internet Inter-Domain Traffic". In: *SIGCOMM Comput. Commun. Rev.* 40.4 (Aug. 2010), pp. 75–86. ISSN: 0146-4833 (cit. on pp. 9, 11).

[15] *Internet Service Provider 3-Tier Model*. https://www.thousandeyes.com/learning/techtorials/isp-tiers. Accessed: 2023-01-08 (cit. on p. 9).

[16] Nikolaos Chatzis, Georgios Smaragdakis, and Anja Feldmann. *On the importance of Internet eXchange Points for today's Internet ecosystem*. 2013 (cit. on p. 10).

[17] *What is an Internet Exchange Point?* https://www.cloudflare.com/learning/cdn/glossary/internet-exchange-point-ixp/. Accessed: 2023-01-08 (cit. on p. 10).

[18] Artur Ziviani. "Internet measurements". In: *Encyclopedia of Internet Technologies and Applications*. IGI Global, 2008, pp. 235–241 (cit. on pp. 12, 14).

[19] The ZMap Team. *ZMap: The Internet Scanner*. https://github.com/zmap/zmap. GitHub repository, Accessed: 28-September-2022. Apr. 2022 (cit. on pp. 14, 68, 87).

[20] Chair of Network Architectures and Services at TUM. *ZMapv6: Internet Scanner with IPv6 capabilities*. https://github.com/tumi8/zmap. GitHub repository, Accessed: 26-October-2022. July 2022 (cit. on pp. 14, 68, 87).

[21] The ZMap Team. *ZGrab 2.0*. https://github.com/zmap/zgrab2. GitHub repository, Accessed: 28-September-2022. July 2022 (cit. on pp. 14, 84, 87).

[22] Yarrp authors. *Yarrp on Github*. 2020 (cit. on pp. 14, 68, 78).

[23] O. Gasser, Q. Scheitle, P. Foremski, Q. Lone, M. Korczynski, S. D. Strowes, L. Hendriks, and G. Carle. "Clusters in the Expanse: Understanding and Unbiasing IPv6 Hitlists". In: *ACM IMC*. 2018 (cit. on pp. 14, 68, 87, 105).

[24] C. Partridge and M. Allman. "Ethical Considerations in Network Measurement Papers". In: *Communications of the ACM* (2016) (cit. on p. 15).

[25] D. Dittrich, E. Kenneally, et al. “The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research”. In: *U.S. Department of Homeland Security* (2012) (cit. on p. 15).

[26] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. “ZMap: Fast Internet-wide Scanning and Its Security Applications”. In: *Proceedings of the 22nd USENIX Security Symposium*. 2013, pp. 605–620 (cit. on pp. 15, 68).

[27] R. Hofstede, P. Celeda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras. “Flow Monitoring Explained: From Packet Captureto Data Analysis With NetFlow and IPFIX”. In: *IEEE Communications Surveys & Tutorials* 16.4 (2014) (cit. on p. 17).

[28] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft. “Structural Analysis of Network Traffic Flows”. In: *ACM SIGMETRICS*. 2005 (cit. on p. 17).

[29] A. Lakhina, M. Crovella, and C. Diot. “Diagnosing Network-Wide Traffic Anomalies”. In: *ACM SIGCOMM*. 2004 (cit. on p. 17).

[30] C. Estan and G. Varghese. “New Directions in Traffic Measurement and Accounting”. In: *ACM SIGCOMM*. 2002 (cit. on pp. 17, 21).

[31] C. Estan, K. Keys, D. Moore, and G. Varghese. “Building a better NetFlow”. In: *ACM SIGCOMM*. 2004 (cit. on pp. 17, 21).

[32] C. Estan, S. Savage, and G. Varghese. “Automatically Inferring Patterns of Resource Consumption in Network Traffic”. In: *ACM SIGCOMM*. 2003 (cit. on p. 17).

[33] *TCPDUMP/LIBPCAP public repository*. `http://www.tcpdump.org/`. 2019 (cit. on p. 17).

[34] N. Duffield, C. Lund, and M. Thorup. “Estimating Flow Distributions from Sampled Flow Statistics”. In: *ACM SIGCOMM*. 2003 (cit. on pp. 17, 18).

[35] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger. “Sonata: Query-driven Streaming Network Telemetry”. In: *ACM SIGCOMM*. 2018 (cit. on pp. 17, 19, 20, 42).

[36] O. Tilmans, T. Bühler, I. Poese, S. Vissicchio, and L. Vanbever. “Stroboscope: Declarative Network Monitoring on a Budget”. In: *NSDI* (2018) (cit. on pp. 17, 19, 20).

[37] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim. “Language-Directed Hardware Design forNetwork Performance Monitoring”. In: *ACM SIGCOMM*. 2017 (cit. on pp. 17, 19, 20).

[38] M. Zaharia and M. Chowdhury and M. J. Franklin and S. Shenker and I. Stoica. “Spark: Cluster Computing with Working Sets”. In: *USENIX HotCloud*. 2010 (cit. on pp. 18, 20, 35, 40).

[39] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. “Gigascope: A Stream Database for Network Applications”. In: *ACM SIGMOD*. 2003 (cit. on pp. 19, 20, 24).

[40] D. Sarlis, N. Papailiou, I. Konstantinou, G. Smaragdakis, and N. Koziris. "Datix: A system for scalable network analytics". In: *ACM CCR* 45.5 (2015) (cit. on p. 19).

[41] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. "Finding hierarchical heavy hitters in data streams". In: *VLDB*. 2003 (cit. on p. 19).

[42] R. B. Basat, G. Einziger, R. Friedman, M. C. Luizelli, and E. Waisbard. "Constant Time Updates in Hierarchical Heavy Hitters". In: *ACM SIGCOMM*. 2017 (cit. on pp. 19, 21, 27).

[43] A. R Curtis, J. C Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. "DevoFlow: scaling flow management for high-performance networks". In: *ACM CCR*. Vol. 41. 4. 2011 (cit. on p. 19).

[44] Q. Huang, X. Jin, P. PC Lee, R. Li, L. Tang, Y. C. Chen, and G. Zhang. "Sketchvisor: Robust network measurement for software packet processing". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 2017 (cit. on pp. 19, 21).

[45] R. Ben Basat, G. Einziger, R. Friedman, and Y. Kassner. "Optimal elephant flow detection". In: *IEEE INFOCOM*. 2017 (cit. on p. 19).

[46] Y. Da, Z. Yibo, A. Behnaz, F. Rodrigo, Z. Tianrong, D. Karl, and Y. Lihua. "dShark: A General, Easy to Program and Scalable Framework for Analyzing In-network Packet Traces". In: *NSDI*. 2019 (cit. on pp. 19, 20).

[47] G. Cormode and S. Muthukrishnan. "An improved data stream summary: The count-min sketch and its applications". In: *Latin American Symposium on Theoretical Informatics*. Springer. 2004, pp. 29–38 (cit. on p. 19).

[48] G. Cormode and M. Hadjieleftheriou. "Finding Frequent Items in Data Streams". In: *VLDB*. 2008 (cit. on p. 19).

[49] G. Cormode and S. Muthukrishnan. "Space Efficient Mining of Multigraph Streams". In: *PODS*. 2005 (cit. on p. 19).

[50] S. Narayana, M. Tahmasbi, J. Rexford, and D. Walker. "Compiling Path Queries". In: *NSDI*. 2016 (cit. on p. 19).

[51] Y. Li, R. Miao, C. Kim, and M. Yu. "FlowRadar: A Better NetFlow for Data Centers." In: *Nsdi*. 2016, pp. 311–324 (cit. on p. 19).

[52] Q. Huang, P. PC Lee, and Y. Bao. "Sketchlearn: relieving user burdens in approximate measurement with automated statistical inference". In: *ACM SIGCOMM*. 2018 (cit. on p. 19).

[53] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig. "Elastic Sketch: Adaptive and Fast Network-wide Measurements". In: *ACM SIGCOMM*. 2018 (cit. on pp. 19, 21).

[54] M. Yu, L. Jose, and R. Miao. "Software Defined Traffic Measurement with OpenSketch". In: *NSDI*. 2013 (cit. on pp. 19, 21).

[55] V. Bajpai and J. Schönwälder. "Network flow query language—Design, implementation, performance, and applications". In: *IEEE TNSM* 14.1 (2016) (cit. on p. 19).

[56] A. G. Prieto and R. Stadler. "A-GAP: An adaptive protocol for continuous network monitoring with accuracy objectives". In: *IEEE TNSM* 4.1 (2007) (cit. on pp. 19, 20).

[57] J. Shuyuan and D. S. Yeung. "A covariance analysis model for DDoS attack detection". In: *IEEE ICC*. 2004 (cit. on p. 19).

[58] V. Sekar, N. G Duffield, O. Spatscheck, J. E van der Merwe, and H. Zhang. "LADS: Large-scale Automated DDoS Detection System." In: *USENIX Annual Technical Conference, General Track*. 2006, pp. 171–184 (cit. on p. 19).

[59] S. M. Mousavi and M. St-Hilaire. "Early detection of DDoS attacks against SDN controllers". In: *ICNC*. 2015 (cit. on p. 19).

[60] A. Metwally, D. Agrawal, and A. El Abbadi. "Efficient computation of frequent and top-k elements in data streams". In: *ICDT*. 2005 (cit. on p. 19).

[61] R. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. A Dinda, M. Y. Kao, and G. Memik. "Reversible sketches: enabling monitoring and analysis over high-speed data streams". In: *IEEE/ACM Trans. Networking* 15.5 (2007) (cit. on p. 19).

[62] L. Tang, Q. Huang, and P. PC Lee. "MV-Sketch: A Fast and Compact Invertible Sketch for Heavy Flow Detection in Network Data Streams". In: *IEEE INFOCOM*. 2019 (cit. on p. 19).

[63] C. Graham and S. Muthukrishnan. "What's new: Finding significant differences in network data streams". In: *IEEE INFOCOM*. 2004 (cit. on p. 19).

[64] P. Tammana, R. Agarwal, and M. Lee. "Simplifying datacenter network debugging with pathdump". In: *ACM OSDI*. 2016 (cit. on p. 19).

[65] P. Tammana, R. Agarwal, and M. Lee. "Distributed Network Monitoring and Debugging with SwitchPointer". In: *NSDI*. 2018 (cit. on p. 19).

[66] B. Arzani, S. Ciraci, L. Chamon, Y. Zhu, H. Liu, J. Padhye, B.T. Loo, and G. Outhred. "007: Democratically Finding the Cause of Packet Drops". In: *NSDI*. 2018 (cit. on p. 19).

[67] *Flowyager in Github*. `https://github.com/saidjawad/Flowyager` (cit. on p. 20).

[68] C. Labovitz, S. Lekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. "Internet Inter-Domain Traffic". In: *ACM SIGCOMM*. 2010 (cit. on p. 20).

[69] R. Caceres, N. Duffield, A. Feldmann, J. D. Friedmann, A. Greenberg, R. Greer, T. Johnson, C. R. Kalmanek, B. Krishnamurthy, D. Lavelle, P. P. Mishra, K. K. Ramakrishnan, J. Rexford, F. True, and J. E. van der Merwe. "Measurement and analysis of IP network usage and behavior". In: *IEEE Communications Magazine* 38.5 (2000) (cit. on p. 20).

[70] European Union. *Data protection in the EU, The General Data Protection Regulation (GDPR); Regulation (EU) 2016/679*. `https://ec.europa.eu/info/law/law-topic/data-protection/`. 2018 (cit. on p. 20).

[71] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann. "OFRewind: Enabling Record and Replay Troubleshooting for Networks". In: *Usenix ATC*. 2011 (cit. on p. 20).

[72] A. Wundsam, A. Mehmood, A. Feldmann, and O. Maennel. "Network troubleshooting with mirror vnets". In: *GLOBECOM Workshops*. 2010 (cit. on p. 20).

[73] Ross Teixeira, Rob Harrison, Arpit Gupta, and Jennifer Rexford. "Packetscope: Monitoring the packet lifecycle inside a switch". In: *Proceedings of the Symposium on SDN Research*. 2020, pp. 76–82 (cit. on p. 20).

[74] Muhammad Tirmazi, Ran Ben Basat, Jiaqi Gao, and Minlan Yu. "Cheetah: Accelerating Database Queries with Switch Pruning". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 2407–2422 (cit. on p. 20).

[75] Damu Ding, Marco Savi, Gianni Antichi, and Domenico Siracusa. "An incrementally-deployable P4-enabled architecture for network-wide heavy-hitter detection". In: *IEEE Transactions on Network and Service Management* 17.1 (2020) (cit. on p. 20).

[76] Ran Ben Basat, Xiaoqi Chen, Gil Einziger, and Ori Rottenstreich. "Designing Heavy-Hitter Detection Algorithms for Programmable Switches". In: *IEEE/ACM Transactions on Networking* (2020) (cit. on p. 20).

[77] S. Pontarelli, R. Bifulco, M. Bonola C. Cascone, M. Spaziani, V. Bruschi, Davide Sanvito, G. Siracusano, A. Capone, M. Honda, F. Huici, and G. Bianchi. "Flowblaze: Stateful packet processing in hardware". In: *NSDI*. 2019 (cit. on p. 20).

[78] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu. "Poseidon: Mitigating volumetric ddos attacks with programmable switches". In: *NDSS*. 2020 (cit. on p. 20).

[79] M. Yu. "Network telemetry: towards a top-down approach". In: *ACM CCR* 49.1 (2019) (cit. on p. 20).

[80] Y. Lee and Y. Lee. "Toward Scalable Internet Traffic Measurement and Analysis with Hadoop". In: *ACM CCR* 43.1 (2013) (cit. on p. 20).

[81] Yandex. *Open source distributed column-oriented DBMS ClickHouse*. `https://clickhouse.yandex/`. 2018 (cit. on pp. 20, 35, 40).

[82] A. Vulimiri, C. Curino, B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese. "Global analytics in the face of bandwidth and regulatory constraints". In: *NSDI*. 2015 (cit. on p. 20).

[83] A. Vulimir, C. Curino, B. Godfrey, K. Karanasos, and G. Varghese. "WANalytics: Analytics for a Geo-Distributed Data-Intensive Worl". In: *CIDR*. 2015 (cit. on p. 20).

[84] R. Viswanathan, G. Ananthanarayanan, and A. Akella. "CLARINET: WAN-Aware Optimization for Analytics Queries". In: *NSDI*. 2016 (cit. on p. 20).

[85] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R Ganger, P. B Gibbons, and O. Mutlu. "Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds". In: *NSDI*. 2017 (cit. on p. 20).

[86] Yuzhen Huang, Yingjie Shi, Zheng Zhong, Yihui Feng, James Cheng, Jiwei Li, Haochuan Fan, Chao Li, Tao Guan, and Jingren Zhou. "Yugong: Geo-Distributed data and job placement at scale". In: *Proceedings of the VLDB Endowment* 12.12 (2019), pp. 2155–2169 (cit. on p. 20).

[87] Alessandro D'Alconzo, Idilio Drago, Andrea Morichetta, Marco Mellia, and Pedro Casas. "A survey on big data for network traffic monitoring and analysis". In: *IEEE Transactions on Network and Service Management* 16.3 (2019), pp. 800–813 (cit. on p. 20).

[88] Ran Ben Basat, Xiaoqi Chen, Gil Einziger, Roy Friedman, and Yaron Kassner. "Randomized admission policy for efficient top-k, frequency, and volume estimation". In: *IEEE/ACM Transactions on Networking* 27.4 (2019), pp. 1432–1445 (cit. on p. 21).

[89] Rob Harrison, Shir Landau Feibish, Arpit Gupta, Ross Teixeira, S Muthukrishnan, and Jennifer Rexford. "Carpe Elephants: Seize the Global Heavy Hitters". In: *Proceedings of the Workshop on Secure Programmable Network Infrastructure*. 2020, pp. 15–21 (cit. on p. 21).

[90] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. "Diamond in the Rough: Finding Hierarchical Heavy Hitters in Multi-Dimensional Data". In: *ACM SIGMOD*. 2004 (cit. on pp. 21, 26, 27).

[91] M. Mitzenmacher, T. Steinke, and J. Thaler. "Hierarchical Heavy Hitters with the Space Saving Algorithm". In: *ALENEX*. 2012 (cit. on pp. 21, 27).

[92] G. Cormode and S. Muthukrishnan. "What's new: Finding significant differences in network data streams". In: *IEEE/ACM Trans. Networking* 13.6 (2005) (cit. on p. 21).

[93] Nikita Ivkin, Ran Ben Basat, Zaoxing Liu, Gil Einziger, Roy Friedman, and Vladimir Braverman. "I know what you did last summer: Network monitoring using interval queries". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3.3 (2019) (cit. on p. 21).

[94] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman. "One sketch to rule them all: Rethinking network flow monitoring with univmon". In: *ACM SIGCOMM*. 2016 (cit. on p. 21).

[95] Theophilus Wellem, Yu-Kuen Lai, Chao-Yuan Huang, and Wen-Yaw Chung. "A flexible sketch-based network traffic monitoring infrastructure". In: *IEEE Access* 7 (2019) (cit. on p. 21).

[96] Haibo Wang, Hongli Xu, Liusheng Huang, and Yutong Zhai. "Fast and Accurate Traffic Measurement With Hierarchical Filtering". In: *IEEE Transactions on Parallel and Distributed Systems* 31.10 (2020), pp. 2360–2374 (cit. on p. 21).

[97] T. Repantis, J. Cohen, S. Smith, and J. Wein. "Scaling a Monitoring Infrastructure for the Akamai Network". In: *SIGOPS Oper. Syst. Rev.* 44.3 (2010) (cit. on p. 24).

[98] J. Cohen, T. Repantis, S. McDermott, S. Smith, and J. Wein. "Keeping Track of 70,000+ Servers: The Akamai Query System". In: *USENIX LISA*. 2010 (cit. on p. 24).

[99] J. Wallerich, H. Dreger, A. Feldmann, B. Krishnamurthy, and W. Willinger. "A Methodology for Studying Persistency Aspects of Internet Flows". In: *ACM CCR* 35.2 (Apr. 2005) (cit. on p. 25).

[100] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. "On the characteristics and origins of internet flow rates". In: *ACM CCR*. Vol. 32. 4. 2002 (cit. on p. 25).

[101] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. "Web caching and Zipf-like distributions: Evidence and implications". In: *IEEE INFOCOM*. 1999 (cit. on p. 25).

[102] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. "Finding Hierarchical Heavy Hitters in Streaming Data". In: *ACM Trans. Knowl. Discov. Data* 1.4 (2008) (cit. on pp. 26, 27).

[103] *mongoDB: The database for modern applications*. `https://www.mongodb.com/`. 2019 (cit. on p. 31).

[104] *Apache Thrift*. `https://thrift.apache.org/`. 2019 (cit. on p. 31).

[105] *ANTLR (ANother Tool for Language Recognition)*. `https://www.antlr.org/`. 2019 (cit. on p. 32).

[106] RStudio, Inc. *Easy Web applications in R*. 2013 (cit. on p. 32).

[107] *MAWI Working Group Traffic Archive*. `http://mawi.wide.ad.jp/mawi/`. 2018 (cit. on pp. 35, 67).

[108] S. T. Zargar, J. Joshi, and D. Tipper. "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks". In: *IEEE communications surveys & tutorials* 15.4 (2013), pp. 2046–2069 (cit. on p. 44).

[109] G. Carl, G. Kesidis, R. R Brooks, and S. Rai. "Denial-of-service attack-detection techniques". In: *IEEE Internet computing* 10.1 (2006), pp. 82–89 (cit. on p. 44).

[110] C. Douligeris and A. Mitrokotsa. "DDoS attacks and defense mechanisms: classification and state-of-the-art". In: *Computer Networks* 44.5 (2004), pp. 643–666 (cit. on p. 44).

[111] K. Lee, J. Kim, K. H. Kwon, Y. Han, and S. Kim. "DDoS attack detection method using cluster analysis". In: *Expert systems with applications* 34.3 (2008), pp. 1659–1665 (cit. on p. 44).

[112] J. Czyz, M. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. Bailey, and M. Karir. "Taming the 800 Pound Gorilla: The Rise and Decline of NTP DDoS Attacks". In: *ACM IMC*. 2014 (cit. on p. 44).

[113] M. Jonker, A. King, J. Krupp, C. Rossow, A. Sperotto, and A. Dainotti. "Millions of Targets Under Attack: A Macroscopic Characterization of the DoS Ecosystem". In: *ACM IMC*. 2017 (cit. on pp. 44, 45).

[114] C. Dietzel, G. Smaragdakis, M. Wichtlhuber, and A. Feldmann. "Stellar: network attack mitigation using advanced blackholing". In: *ACM CoNEXT*. 2018 (cit. on pp. 44, 45).

[115] M. Jonker, A. Sperotto, R. van Rijswijk-Deij, R. Sadre, and A. Pras. "Measuring the Adoption of DDoS Protection Services". In: *ACM IMC*. 2016 (cit. on p. 44).

[116] *US-Cert: Alert (TA14-017A), UDP-Based Amplification Attacks*. `https://www.us-cert.gov/ncas/alerts/TA14-017A`. 2019 (cit. on p. 44).

[117] Akamai. *State of the Internet Security Report (Attack Spotlight: Memcached)*. `https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/soti-summer-2018-attack-spotlight.pdf`. 2018 (cit. on p. 44).

[118] C. Rossow. "Amplification Hell: Revisiting Network Protocols for DDoS Abuse". In: *NDSS* (2014) (cit. on p. 44).

[119] Cisco. *Cisco IOS NetFlow*. `https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html`. 2021 (cit. on p. 47).

[120] Paul Aitken, Benoît Claise, and Brian Trammell. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. RFC 7011. Sept. 2013 (cit. on p. 47).

[121] Ernie Regalado. *The Multi-CDN Strategy*. Accessed: 2022-06-27. 2014 (cit. on p. 47).

[122] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. "Measuring {HTTPS} adoption on the web". In: *26th USENIX Security Symposium (USENIX Security 17)*. 2017, pp. 1323–1338 (cit. on p. 47).

[123] P. Mockapetris. *Domain names - concepts and facilities*. RFC 1034. RFC Editor, Nov. 1987, pp. 1–55. URL: `https://www.ietf.org/rfc/rfc1034.txt` (cit. on p. 47).

[124] P. Mockapetris. *Domain names - implementation and specification*. RFC 1035. RFC Editor, Nov. 1987, pp. 1–55. URL: `https://www.ietf.org/rfc/rfc1035.txt` (cit. on pp. 47, 62).

[125] Xunxun Chen, Gaochao Li, Yongzheng Zhang, Xiao Wu, and Changbo Tian. "A Deep Learning Based Fast-Flux and CDN Domain Names Recognition Method". In: *Proceedings of the 2019 2nd International Conference on Information Science and Systems*. ICISS 2019. Tokyo, Japan: Association for Computing Machinery, 2019, pp. 54–59. ISBN: 9781450361033 (cit. on p. 47).

[126] Hailing Li, Longtao He, Hui Zhang, Kai Zhang, Xiaoqian Li, and Chenghai He. "CDN-Hosted Domain Detection with Supervised Machine Learning through DNS Records". In: *Proceedings of the 2020 The 3rd International Conference on Information Science and System*. ICISS 2020. Cambridge, United Kingdom: Association for Computing Machinery, 2020, pp. 144–149. ISBN: 9781450377256 (cit. on p. 47).

[127] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. "Exposure: Finding malicious domains using passive DNS analysis." In: *The Network and Distributed System Security (NDSS) Symposium*. 2011, pp. 1–17 (cit. on p. 47).

[128] Zhouyu Bao, Wenbo Wang, and Yuqing Lan. "Using Passive DNS to Detect Malicious Domain Name". In: *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing*. ICVISP 2019. Vancouver, BC, Canada: Association for Computing Machinery, 2019. ISBN: 9781450376259 (cit. on p. 47).

[129] Roberto Perdisci, Thomas Papastergiou, Omar Alrawi, and Manos Antonakakis. "Iotfinder: Efficient large-scale identification of iot devices via passive dns traffic analysis". In: *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE Computer Society. 2020, pp. 474–489 (cit. on p. 47).

[130] Sivaramakrishnan Ramanathan, Anushah Hossain, Jelena Mirkovic, Minlan Yu, and Sadia Afroz. "Quantifying the Impact of Blocklisting in the Age of Address Reuse". In: *Proceedings of the ACM Internet Measurement Conference*. IMC '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 360–369. ISBN: 9781450381383 (cit. on p. 47).

[131] Ramakrishna Padmanabhan, John P Rula, Philipp Richter, Stephen D Strowes, and Alberto Dainotti. "DynamIPs: analyzing address assignment practices in IPv4 and IPv6". In: *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*. 2020, pp. 55–70 (cit. on p. 47).

[132] Sean Donovan and Nick Feamster. "Intentional network monitoring: Finding the needle without capturing the haystack". In: *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*. 2014, pp. 1–7 (cit. on p. 47).

[133] Jason Kim, Hyojoon Kim, and Jennifer Rexford. "Analyzing traffic by domain name in the data plane". In: *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*. 2021, pp. 1–12 (cit. on p. 47).

[134] Ali AlSabeh, Elie Kfoury, Jorge Crichigno, and Elias Bou-Harb. "P4DDPI: Securing P4-Programmable Data Plane Networks via DNS Deep Packet Inspection". In: *Proceedings of the 2022 Network and Distributed System Security (NDSS) Symposium*. 2022, pp. 1–7 (cit. on p. 47).

[135] Giovane C. M. Moura, John Heidemann, Ricardo de O. Schmidt, and Wes Hardaker. "Cache Me If You Can: Effects of DNS Time-to-Live". In: *Proceedings of the Internet Measurement Conference*. IMC '19. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 101–115. ISBN: 9781450369480 (cit. on p. 48).

[136] Audrey Randall, Enze Liu, Gautam Akiwate, Ramakrishna Padmanabhan, Geoffrey M Voelker, Stefan Savage, and Aaron Schulman. "Trufflehunter: cache snooping rare domains at large public DNS resolvers". In: *Proceedings of the ACM Internet Measurement Conference*. 2020, pp. 50–64 (cit. on p. 48).

[137] Orcaman. *A Thread-Safe Concurrent Map for Go*. Accessed: 2022-10-16. 2022. URL: `https://github.com/orcaman/concurrent-map` (cit. on p. 50).

[138] Digineo GmbH. *Public DNS Server List*. Accessed: 2022-10-05. 2022. URL: `https://public-dns.info/` (cit. on p. 58).

[139] Gopinath Palaniappan, Sangeetha S, Balaji Rajendran, Sanjay, Shubham Goyal, and Bindhumadhava B S. "Malicious Domain Detection Using Machine Learning On Domain Name Features, Host-Based Features and Web-Based Features". In: *Procedia Computer Science* 171 (2020). Third International Conference on Computing and Network Communications (CoCoNet'19), pp. 654–661. ISSN: 1877-0509 (cit. on p. 60).

[140] Sara Afzal, Muhammad Asim, Abdul Rehman Javed, Mirza Omer Beg, and Thar Baker. "Urldeepdetect: A deep learning approach for detecting malicious urls using semantic vector models". In: *Journal of Network and Systems Management* 29.3 (2021), pp. 1–27 (cit. on p. 60).

[141] Sandeep Yadav, Ashwath Kumar Krishna Reddy, A.L. Narasimha Reddy, and Supranamaya Ranjan. "Detecting Algorithmically Generated Malicious Domain Names". In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC '10. Melbourne, Australia: Association for Computing Machinery, 2010, pp. 48–61. ISBN: 9781450304832 (cit. on p. 60).

[142] Xiaoqing Sun, Mingkai Tong, Jiahai Yang, Liu Xinran, and Liu Heng. "HinDom: A Robust Malicious Domain Detection System based on Heterogeneous Information Network with Transductive Classification". In: *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. Chaoyang District, Beijing: USENIX Association, Sept. 2019, pp. 399–412. ISBN: 978-1-939133-07-6 (cit. on p. 60).

[143] Ramin Yazdani, Olivier van der Toorn, and Anna Sperotto. "A Case of Identity: Detection of Suspicious IDN Homograph Domains Using Active DNS Measurements". In: *2020 IEEE European Symposium on Security and Privacy Workshops (EuroSPW)*. 2020, pp. 559–564 (cit. on p. 60).

[144] Zicong Zhu, Tran Phuong Thao, Hoang-Quoc Nguyen-Son, Rie Shigetomi Yamaguchi, and Toshiyuki Nakata. "Enhancing A New Classification for IDN Homograph Attack Detection". In: *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing*. 2020, pp. 507–514 (cit. on p. 60).

[145] Hiroaki Suzuki, Daiki Chiba, Yoshiro Yoneya, Tatsuya Mori, and Shigeki Goto. "ShamFinder: An Automated Framework for Detecting IDN Homographs". In: *Proceedings of the Internet Measurement Conference*. IMC '19. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 449–462. ISBN: 9781450369480 (cit. on p. 60).

[146] Pelayo Vallina, Victor Le Pochat, Álvaro Feal, Marius Paraschiv, Julien Gamba, Tim Burke, Oliver Hohlfeld, Juan Tapiador, and Narseo Vallina-Rodriguez. "Mis-Shapes, Mistakes, Misfits: An Analysis of Domain Classification Services". In: *Proceedings of the ACM Internet Measurement Conference*. IMC '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 598–618. ISBN: 9781450381383 (cit. on p. 60).

[147] Baojun Liu, Chaoyi Lu, Zhou Li, Ying Liu, Haixin Duan, Shuang Hao, and Zaifeng Zhang. "A Reexamination of Internationalized Domain Names: The Good, the Bad and the Ugly". In: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2018, pp. 654–665 (cit. on p. 61).

[148] Unicode. *Recommended confusable mapping for IDN*. Accessed: 2022-06-12. 2015. URL: https://www.unicode.org/Public/security/8.0.0/confusables.txt (cit. on p. 61).

[149] Spamhaus Project. *Spamhaus DBL*. https://www.spamhaus.org/dbl/. 2022 (cit. on p. 62).

[150] J. Reynolds and J. Postel. *Assigned Numbers*. RFC 1700 (Historic). RFC. Obsoleted by RFC 3232. Fremont, CA, USA: RFC Editor, Oct. 1994. DOI: 10.17487/RFC1700. URL: https://www.rfc-editor.org/rfc/rfc1700.txt (cit. on pp. 65, 66).

[151] L-A. Larzon, M. Degermark, S. Pink, L-E. Jonsson, Ed. Ericsson, G. Fairhurst. *The Lightweight User Datagram Protocol (UDP-Lite)*. RFC 3828. RFC Editor, July 2004 (cit. on pp. 65, 66).

[152] R. Stewart. *Stream Control Transmission Protocol*. RFC 4960. RFC Editor, Sept. 2007 (cit. on pp. 65, 66).

[153] Linux man-pages project. *bind(2) — Linux manual page*. 2020. URL: https://man7.org/linux/man-pages/man2/bind.2.html (cit. on p. 65).

[154] Microsoft. *Windows bind function*. 2018. URL: https://docs.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-bind (cit. on p. 65).

[155] Elias Bou-Harb, Nour-Eddine Lakhdari, Hamad Binsalleeh, and Mourad Debbabi. "Multidimensional investigation of source port 0 probing". In: *Digital Investigation* 11 (2014), S114–S123 (cit. on pp. 65, 66).

[156] Marek Majkowski. *Reflections on reflection (attacks)*. 2017. URL: https://blog.cloudflare.com/reflections-on-reflections/ (cit. on p. 66).

[157] Rick Wanner. *Port 0 DDOS*. 2013. URL: https://isc.sans.edu/forums/diary/Port+0+DDOS/17081/ (cit. on p. 66).

[158] Tom Jones. *DDoS Attacks on Port 0 – Does it mean what you think it does?* 2013. URL: https://blog.endace.com/2013/08/27/ddos-attacks-on-port-0-does-it-mean-what-you-think-it-does/ (cit. on p. 66).

[159] Marina Bykova and Shawn Ostermann. "Statistical Analysis of Malformed Packets and Their Origins in the Modern Internet". In: *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*. IMW '02. Marseille, France: Association for Computing Machinery, 2002, pp. 83–88. ISBN: 158113603X (cit. on p. 66).

[160] Eric Wustrow, Manish Karir, Michael Bailey, Farnam Jahanian, and Geoff Huston. "Internet Background Radiation Revisited". In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC '10. Melbourne, Australia: Association for Computing Machinery, 2010, pp. 62–74. ISBN: 9781450304832 (cit. on pp. 66, 73, 74).

[161] WIDE project. *WIDE project website*. 2020. URL: `http://www.wide.ad.jp/index_e.html` (cit. on p. 67).

[162] CAIDA. *A Day in the Life of the Internet (DITL)*. 2020. URL: `https://www.caida.org/projects/ditl/` (cit. on p. 67).

[163] CAIDA. *The CAIDA Anonymized Internet Traces Data Access*. 2019. URL: `https://www.caida.org/data/passive/passive_dataset_download.xml` (cit. on p. 67).

[164] WAND Network Research Group. *WITS: Waikato VIII*. 2020. URL: `https://wand.net.nz/wits/waikato/8/` (cit. on p. 67).

[165] O. Gasser, Q. Scheitle, S. Gebhard, and G. Carle. "Scanning the IPv6 Internet: Towards a Comprehensive Hitlist". In: *TMA*. 2016 (cit. on p. 68).

[166] Oliver Gasser, Quirin Scheitle, Pawel Foremski, Qasim Lone, Maciej Korczynski, Stephen D. Strowes, Luuk Hendriks, and Georg Carle. *IPv6 Hitlist Service*. 2018. URL: `https://ipv6hitlist.github.io/` (cit. on pp. 68, 76).

[167] Robert Beverly. "Yarrp'ing the Internet: Randomized High-SpeedActive Topology Discovery". In: *Proceedings of the Internet Measurement Conference*. 2016, pp. 413–420 (cit. on p. 68).

[168] ACM. *Artifact Review and Badging*. 2020. URL: `https://www.acm.org/publications/policies/artifact-review-badging` (cit. on p. 68).

[169] Quirin Scheitle, Matthias Wählisch, Oliver Gasser, Thomas C. Schmidt, and Georg Carle. "Towards an Ecosystem for Reproducible Research in Computer Networking". In: *Proceedings of the ACM SIGCOMM Reproducibility Workshop*. 2017 (cit. on p. 68).

[170] Oliver Gasser. *Analysis scripts and raw data for active port 0 measurements*. 2021. DOI: `10.17617/3.5f` (cit. on p. 68).

[171] Daniel Kopp, Christoph Dietzel, and Oliver Hohlfeld. "DDoS Never Dies? An IXP Perspective on DDoS Amplification Attacks". In: *Proceedings of the Passive and Active Measurement Conference*. 2021 (cit. on p. 69).

[172] Nokia. *Router Configuration Guide Release 16.0.R4*. `https://infoproducts.nokia.com/cgi-bin/dbaccessfilename.cgi/3HE14136AAABTQZZA01_V1_7450`. 2018 (cit. on p. 69).

[173] Paul Aitken. *RFC Erratum 1738*. 2009 (cit. on p. 69).

[174] Shane Alcock and Richard Nelson. "Libprotoident: Traffic Classification UsingLightweight Packet Inspection". In: *WAND Network Research Group, Tech. Rep* (2012) (cit. on p. 72).

[175] Roger Hallman, Josiah Bryan, Geancarlo Palavicini, Joseph Divita, and Jose Romero-Mariona. "IoDDoS-the internet of distributed denial of sevice attacks". In: *2nd international conference on internet of things, big data and security. SCITEPRESS*. 2017, pp. 47–58 (cit. on p. 74).

[176] Hadi Asghari. *pyasn on Github*. 2018. URL: %5Curl%7Bhttps://github.com/hadiasghari/pyasn%7D (cit. on p. 77).

[177] CAIDA. *Routeviews Prefix-to-AS mappings (pfx2as) for IPv4 and IPv6*. 2020. URL: http://data.caida.org/datasets/routing/routeviews-prefix2as/ (cit. on p. 77).

[178] Ayman Mukaddam, Imad Elhajj, Ayman Kayssi, and Ali Chehab. "IP Spoofing Detection Using Modified Hop Count". In: *Proceedings of the Advanced Information Networking and Applications Conference*. 2014 (cit. on p. 77).

[179] Cheng Jin, Haining Wang, and Kang G Shin. "Hop-Count Filtering: An Effective Defense Against Spoofed DDoS Traffic". In: *Proceedings of the ACM Computer and Communications Security Conference*. 2003 (cit. on p. 77).

[180] Michael Backes, Thorsten Holz, Christian Rossow, Teemu Rytilahti, Milivoj Simeonovski, and Ben Stock. "On the Feasibility of TTL-based Filtering for DRDoS Mitigation". In: *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses*. 2016 (cit. on p. 77).

[181] Douglas Fischer. *nanog mailing list: TCP and UDP Port 0 - Should an ISP or ITP Block it?* 2020. URL: https://mailman.nanog.org/pipermail/nanog/2020-August/209228.html (cit. on p. 78).

[182] Xfinity. *Blocked Internet Ports List*. 2020. URL: https://www.xfinity.com/support/articles/list-of-blocked-ports (cit. on p. 78).

[183] AT&T. *Broadband Information - Network Practices*. 2020. URL: https://about.att.com/sites/broadband/network (cit. on p. 78).

[184] Jakub Czyz, Matthew Luckie, Mark Allman, Michael Bailey, et al. "Don't Forget to Lock the Back Door! A Characterization of IPv6 Network Security Policy". In: *Proceedings of the Network and Distributed Systems Security Symposium*. 2016 (cit. on p. 79).

[185] Timm Böttger, Ghida Ibrahim, and Ben Vallis. "How the Internet Reacted to Covid-19: A Perspective from Facebook's Edge Network". In: *Proceedings of the ACM Internet Measurement Conference*. IMC '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 34–41. ISBN: 9781450381383 (cit. on p. 83).

[186] Shinan Liu, Paul Schmitt, Francesco Bronzino, and Nick Feamster. "Characterizing Service Provider Response to the COVID-19 Pandemic in the United States". In: *Passive and Active Measurement*. Ed. by Oliver Hohlfeld, Andra Lutu, and Dave Levin. Cham: Springer International Publishing, 2021, pp. 20–38 (cit. on p. 83).

[187] Mehdi Karamollahi, Carey Williamson, and Martin Arlitt. "Zoomiversity: A Case Study of Pandemic Effects on Post-secondary Teaching and Learning". In: *Passive and Active Measurement*. Ed. by Oliver Hohlfeld, Giovane Moura, and Cristel Pelsser. Cham: Springer International Publishing, 2022, pp. 573–599 (cit. on p. 83).

[188] Matthew Haag. "Remote work is here to stay. Manhattan may never be the same". In: *The New York Times* (2021). URL: https://www.nytimes.com/2021/03/29/nyregion/remote-work-coronavirus-pandemic.html (cit. on p. 83).

[189] Brian Robinson. "Remote Work Is Here To Stay And Will Increase Into 2023, Experts Say". In: *Forbes* (Feb. 2022). URL: https://www.forbes.com/sites/bryanrobinson/2022/02/01/remote-work-is-here-to-stay-and-will-increase-into-2023-experts-say/ (cit. on p. 83).

[190] Pijus Jauniskis. *VPN statistics: Users, markets, & legality*. https://surfshark.com/blog/vpn-users. Accessed: 10-October-2022. Mar. 2022 (cit. on p. 83).

[191] Marzieh Bitaab, Haehyun Cho, Adam Oest, Penghui Zhang, Zhibo Sun, Rana Pourmohamad, Doowon Kim, Tiffany Bao, Ruoyu Wang, Yan Shoshitaishvili, Adam Doupé, and Gail-Joon Ahn. "Scam Pandemic: How Attackers Exploit Public Fear through Phishing". In: *2020 APWG Symposium on Electronic Crime Research (eCrime)*. 2020, pp. 1–10 (cit. on p. 84).

[192] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. "Characterization of encrypted and vpn traffic using time-related". In: *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*. 2016, pp. 407–414 (cit. on p. 84).

[193] Shane Miller, Kevin Curran, and Tom Lunney. "Detection of Virtual Private Network Traffic Using Machine Learning". In: *International Journal of Wireless Networks and Broadband Technologies (IJWNBT)* 9.2 (2020), pp. 60–80 (cit. on p. 84).

[194] Muhammad Zain ul Abideen, Shahzad Saleem, and Madiha Ejaz. "VPN traffic detection in ssl-protected channel". In: *Security and Communication Networks* 2019 (2019) (cit. on p. 84).

[195] Mohammad Taha Khan, Joe DeBlasio, Geoffrey M. Voelker, Alex C. Snoeren, Chris Kanich, and Narseo Vallina-Rodriguez. "An Empirical Analysis of the Commercial VPN Ecosystem". In: *Proceedings of the Internet Measurement Conference 2018*. IMC '18. Boston, MA, USA: Association for Computing Machinery, 2018, pp. 443–456. ISBN: 9781450356190 (cit. on p. 84).

[196] Reethika Ramesh, Leonid Evdokimov, Diwen Xue, and Roya Ensafi. "VPNalyzer: Systematic Investigation of the VPN Ecosystem". In: *Network and Distributed System Security*. The Internet Society, 2022 (cit. on pp. 84, 90, 98, 106).

[197] Lukas Vermeulen. *ZGrab2 VPN modules on GitHub*. `https://github.com/vpnecosystem/zgrab2-vpn` (cit. on pp. 84, 105).

[198] David Crawshaw. "Everything VPN is New Again: The 24-Year-Old Security Model Has Found a Second Wind." In: *Queue* 18.5 (Oct. 2020), pp. 54–66. ISSN: 1542-7730 (cit. on pp. 85, 86, 106).

[199] Agnieszka Dutkowska-Żuk, Austin Hounsel, Amy Morrill, Andre Xiong, Marshini Chetty, and Nick Feamster. "How and Why People Use Virtual Private Networks". In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 3451–3465. ISBN: 978-1-939133-31-1 (cit. on p. 85).

[200] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, and G. Zorn. *Point-to-Point Tunneling Protocol (PPTP)*. RFC 2637 (Informational). RFC. Fremont, CA, USA: RFC Editor, July 1999. URL: `https://www.rfc-editor.org/rfc/rfc2637.txt` (cit. on p. 86).

[201] Microsoft. *Microsoft Security Advisory 2743314*. `https://learn.microsoft.com/en-us/security-updates/SecurityAdvisories/2012/2743314`. [Accessed: 26-October-2022]. 2012 (cit. on p. 86).

[202] Johannes Zirngibl, Lion Steger, Patrick Sattler, Oliver Gasser, and Georg Carle. "Rusty Clusters? Dusting an IPv6 Research Foundation". In: *Proceedings of the 2022 Internet Measurement Conference*. Nice, France: ACM, 2022 (cit. on p. 87).

[203] OpenVPN. *Deprecated Options in OpenVPN*. `https://community.openvpn.net/openvpn/wiki/DeprecatedOptions#Option:--key-method`. [Accessed: 26-October-2022] (cit. on p. 87).

[204] Synopsis, Inc. *The Heartbleed Bug*. `https://heartbleed.com/`. [Online; accessed 29-July-2022]. 2020 (cit. on pp. 88, 96).

[205] R. Seggelmann, M. Tuexen, and M. Williams. *Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension*. RFC 6520 (Proposed Standard). RFC. Updated by RFC 8447. Fremont, CA, USA: RFC Editor, Feb. 2012 (cit. on p. 88).

[206] Gordon Lyon. *Nmap*. `https://nmap.org/`. [Accessed: 26-October-2022] (cit. on p. 88).

[207] The MaxMind Company. *GeoLite2 Free Geolocation Data*. `https://dev.maxmind.com/geoip/geolite2-free-geolocation-data`. Accessed: 06-October-2022. Oct. 2022 (cit. on p. 91).

[208] Nadhem AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt. "On the Security of RC4 in TLS". In: *22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C.: USENIX Association, Aug. 2013, pp. 305–320. ISBN: 978-1-931971-03-4. URL: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/alFardan (cit. on p. 96).

[209] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. *This POODLE Bites: Exploiting The SSL 3.0 Fallback*. https://www.openssl.org/~bodo/ssl-poodle.pdf. Accessed: 19-October-2022. Sept. 2014 (cit. on p. 96).

[210] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. *Tracking the FREAK Attack*. https://freakattack.com/. [Accessed: 19-October-2022]. 2015 (cit. on p. 96).

[211] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. "Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice". In: *22nd ACM Conference on Computer and Communications Security*. Oct. 2015 (cit. on p. 96).

[212] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. "DROWN: Breaking TLS with SSLv2". In: *25th USENIX Security Symposium*. Aug. 2016 (cit. on p. 96).

[213] Hanno Böck, Juraj Somorovsky, and Craig Young. "Return Of Bleichenbacher's Oracle Threat (ROBOT)". In: *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 817–849. ISBN: 978-1-939133-04-5. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/bock (cit. on p. 96).

[214] Robert Merget, Marcus Brinkmann, Nimrod Aviram, Juraj Somorovsky, Johannes Mittmann, and Jörg Schwenk. "Raccoon Attack: Finding and Exploiting Most-Significant-Bit-Oracles in TLS-DH(E)". In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 213–230. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/merget (cit. on p. 96).

[215] Daniel R. Thomas, Alastair R. Beresford, and Andrew Rice. "Security Metrics for the Android Ecosystem". In: *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. SPSM '15. Denver, Colorado, USA: Association for Computing Machinery, 2015, pp. 87–98. ISBN: 9781450338196 (cit. on p. 97).

[216] Kailani R. Jones, Ting-Fang Yen, Sathya Chandran Sundaramurthy, and Alexandru G. Bardas. "Deploying Android Security Updates: An Extensive Study Involving Manufacturers, Carriers, and End Users". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS

'20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 551–567. ISBN: 9781450370899 (cit. on p. 97).

[217] Michael Horowitz. *TCP Ports to Test*. `https://routersecurity.org/testrouter.php#TCPports`. [Online; accessed 13-October-2022] (cit. on p. 98).

[218] PyPi. *Public Suffix PyPi*. `https://pypi.org/project/publicsuffix/`. Accessed: 12-October-2022. Oct. 2022 (cit. on p. 102).

[219] OpenVPN. *Typical Network Configuration*. `https://openvpn.net/access-server-manual/typical-network-configurations/`. [Accessed: 28-October-2022] (cit. on p. 103).

[220] Internet Assigned Numbers Authority. *Service Name and Transport Protocol Port Number Registry*. `https://www.iana.org/assignments/service-names-port-numbers`. Accessed: 25-October-2022. Oct. 2022 (cit. on p. 105).

[221] Liz Izhikevich, Renata Teixeira, and Zakir Durumeric. "Predicting IPv4 Services across All Ports". In: *Proceedings of the ACM SIGCOMM 2022 Conference*. SIGCOMM '22. Amsterdam, Netherlands: Association for Computing Machinery, 2022, pp. 503–515. ISBN: 9781450394208 (cit. on pp. 105, 112).

[222] Oliver Gasser. "Analysis scripts and raw data for VPN ecosystem measurements". In: (2023). URL: `https://doi.org/10.17617/3.NZUPN4` (cit. on p. 105).

[223] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. "Deep packet: A novel approach for encrypted traffic classification using deep learning". In: *Soft Computing* 24.3 (2020), pp. 1999–2012 (cit. on p. 105).

[224] Zhuang Zou, Jingguo Ge, Hongbo Zheng, Yulei Wu, Chunjing Han, and Zhongjiang Yao. "Encrypted Traffic Classification with a Convolutional Long Short-Term Memory Neural Network". In: *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. 2018, pp. 329–334 (cit. on p. 105).

[225] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. "End-to-end encrypted traffic classification with one dimensional convolution neural networks". In: *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*. 2017, pp. 43–48 (cit. on p. 105).

[226] Mustafa Al-Fayoumi, Mohammad Al-Fawa'reh, and Shadi Nashwan. "VPN and Non-VPN Network Traffic Classification Using Time-Related Features". In: *Computers, Materials and Continua* 72 (Mar. 2022), pp. 3091–3111 (cit. on p. 105).

[227] Aniss Maghsoudlou, Oliver Gasser, and Anja Feldmann. "Zeroing in on Port 0 Traffic in the Wild". In: *International Conference on Passive and Active Network Measurement*. Springer. 2021, pp. 547–563 (cit. on p. 106).

[228] Chaoyi Lu, Baojun Liu, Zhou Li, Shuang Hao, Haixin Duan, Mingming Zhang, Chunying Leng, Ying Liu, Zaifeng Zhang, and Jianping Wu. "An End-to-End, Large-Scale Measurement of DNS-over-Encryption: How Far Have We Come?" In: *Proceedings of the Internet Measurement Conference*. IMC '19. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 22–35. ISBN: 9781450369480 (cit. on p. 106).

[229] "OpenVPN is Open to VPN Fingerprinting". In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022. URL: %5Curl%7Bhttps://www.usenix.org/conference/usenixsecurity22/presentation/xue-diwen%7D (cit. on p. 106).

[230] William J. Tolley, Beau Kujath, Mohammad Taha Khan, Narseo Vallina-Rodriguez, and Jedidiah R. Crandall. "Blind In/On-Path Attacks and Applications to VPNs". In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 3129–3146. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/tolley (cit. on p. 106).

[231] Jason Donenfeld. *WireGuard: Next Generation Kernel Network Tunnel*. Tech. rep. Jan. 2017 (cit. on p. 106).

[232] Jacob Appelbaum, Marsh Ray, Karl Koscher, and Ian Finder. "vpwns: Virtual pwned networks". In: *2nd USENIX Workshop on Free and Open Communications on the Internet. USENIX Association*. 2012 (cit. on p. 106).

[233] Ram Sundara Raman, Leonid Evdokimov, Eric Wurstrow, J Alex Halderman, and Roya Ensafi. "Investigating large scale HTTPS interception in Kazakhstan". In: *Proceedings of the ACM Internet Measurement Conference*. 2020, pp. 125–132 (cit. on p. 106).

[234] Muhammad Ikram, Narseo Vallina-Rodriguez, Suranga Seneviratne, Mohamed Ali Kaafar, and Vern Paxson. "An analysis of the privacy and security risks of android vpn permission-enabled apps". In: *Proceedings of the 2016 internet measurement conference*. 2016, pp. 349–364 (cit. on p. 106).

[235] Sangtae Lee, Youngjoo Shin, and Junbeom Hur. "Return of version downgrade attack in the era of TLS 1.3". In: *Proceedings of the 16th International Conference on Emerging Networking Experiments and Technologies*. 2020, pp. 157–168 (cit. on p. 106).

List of Figures

List of Tables