UNIVERSITÄT DES SAARLANDES

Universität des Saarlandes

Characterizing the IoT Ecosystem at Scale

A dissertation submitted towards the degree
Doctor of Natural Sciences
of the Faculty of Mathematics and Computer Science
of Saarland University

by

Said Jawad Saidi

Saarbrücken, 2022

Day of Colloquium	24 / 02 / 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. Georgios Smaragdakis
	Prof. Dr. Jörg Ott
Academic Assistant	Dr. Jialong Li

Abstract

Internet of Things (IoT) devices are extremely popular with home, business, and industrial users. To provide their services, they typically rely on a backend server infrastructure on the Internet, which collectively form the IoT Ecosystem. This ecosystem is rapidly growing and offers users an increasing number of services. It also has been a source and target of significant security and privacy risks. One notable example is the recent large-scale coordinated global attacks, like Mirai, which disrupted large service providers. Thus, characterizing this ecosystem yields insights that help end-users, network operators, policymakers, and researchers better understand it, obtain a detailed view, and keep track of its evolution. In addition, they can use these insights to inform their decision-making process for mitigating this ecosystem's security and privacy risks. In this dissertation, we characterize the IoT ecosystem *at scale* by (i) detecting the IoT devices in the wild, (ii) conducting a case study to measure how deployed IoT devices can affect users' privacy, and (iii) detecting and measuring the IoT backend infrastructure.

To conduct our studies, we collaborated with a large European Internet Service Provider (ISP) and a major European Internet eXchange Point (IXP). They routinely collect *large volumes* of passive, sampled data, e.g., NetFlow and IPFIX, for their operational purposes. These data sources help providers obtain insights about their networks, and we used them to characterize the IoT ecosystem at scale.

We start with IoT devices and study how to track and trace their activity in the wild. We developed and evaluated a scalable methodology to accurately detect and monitor IoT devices with limited, sparsely sampled data in the ISP and IXP.

Next, we conduct a case study to measure how a myriad of deployed devices can affect the privacy of ISP subscribers. Unfortunately, we found that the privacy of a substantial fraction of IPv6 end-users is at risk. We noticed that a single device at home that encodes its MAC address into the IPv6 address could be utilized as a tracking identifier for the entire end-user prefix—even if other devices use IPv6 privacy extensions. Our results showed that IoT devices contribute the most to this privacy leakage.

Finally, we focus on the backend server infrastructure and propose a methodology to identify and locate IoT backend servers operated by cloud services and IoT vendors. We analyzed their IoT traffic patterns as observed in the ISP. Our analysis sheds light on their diverse operational and deployment strategies.

The need for issuing *a priori unknown network-wide* queries against large volumes of network flow capture data, which we used in our studies, motivated us to develop Flowyager. It is a system built on top of existing traffic capture utilities, and it relies on flow summarization techniques to reduce (i) the storage and transfer cost of flow captures and (ii) query response time. We deployed a prototype of Flowyager at both the IXP and ISP.

Zusammenfassung

Internet-of-Things-Geräte (IoT) sind aus vielen Haushalten, Büroräumen und Industrieanlagen nicht mehr wegzudenken. Um ihre Dienste zu erbringen, nutzen IoT-Geräte typischerweise auf eine Backend-Server-Infrastruktur im Internet, welche als Gesamtheit das IoT-Ökosystem bildet. Dieses Ökosystem wächst rapide an und bietet den Nutzern immer mehr Dienste an. Das IoT-Ökosystem ist jedoch sowohl eine Quelle als auch ein Ziel von signifikanten Risiken für die Sicherheit und Privatsphäre. Ein bemerkenswertes Beispiel sind die jüngsten groß angelegten, koordinierten globalen Angriffe wie Mirai, durch die große Diensteanbieter gestört haben. Deshalb ist es wichtig, dieses Ökosystem zu charakterisieren, eine ganzheitliche Sicht zu bekommen und die Entwicklung zu verfolgen, damit Forscher, Entscheidungsträger, Endnutzer und Netzwerkbetreibern Einblicke und ein besseres Verständnis erlangen. Außerdem können alle Teilnehmer des Ökosystems diese Erkenntnisse nutzen, um ihre Entscheidungsprozesse zur Verhinderung von Sicherheits- und Privatsphärerisiken zu verbessern. In dieser Dissertation charakterisieren wir die Gesamtheit des IoT-Ökosystems indem wir (i) IoT-Geräte im Internet detektieren, (ii) eine Fallstudie zum Einfluss von benutzten IoT-Geräten auf die Privatsphäre von Nutzern durchführen und (iii) die IoT-Backend-Infrastruktur aufdecken und vermessen.

Um unsere Studien durchzuführen, arbeiten wir mit einem großen europäischen Internet-Service-Provider (ISP) und einem großen europäischen Internet-Exchange-Point (IXP) zusammen. Diese sammeln routinemäßig für operative Zwecke große Mengen an passiven gesampelten Daten (z.B. als NetFlow oder IPFIX). Diese Datenquellen helfen Netzwerkbetreibern Einblicke in ihre Netzwerke zu erlangen und wir verwendeten sie, um das IoT-Ökosystem ganzheitlich zu charakterisieren.

Wir beginnen unsere Analysen mit IoT-Geräten und untersuchen, wie diese im Internet aufgespürt und verfolgt werden können. Dazu entwickelten und evaluierten wir eine skalierbare Methodik, um IoT-Geräte mit Hilfe von eingeschränkten gesampelten Daten des ISPs und IXPs präzise erkennen und beobachten können.

Als Nächstes führen wir eine Fallstudie durch, in der wir messen, wie eine Unzahl von eingesetzten Geräten die Privatsphäre von ISP-Nutzern beeinflussen kann. Leider fanden wir heraus, dass die Privatsphäre eines substantiellen Teils von IPv6-Endnutzern bedroht ist. Wir entdeckten, dass bereits ein einzelnes Gerät im Haus, welches seine MAC-Adresse in die IPv6-Adresse kodiert, als Tracking-Identifikator für das gesamte Endnutzer-Präfix missbraucht werden kann — auch wenn andere Geräte IPv6-Privacy-Extensions verwenden. Unsere Ergebnisse zeigten, dass IoT-Geräte den Großteil dieses Privatsphäre-Verlusts verursachen.

Abschließend fokussieren wir uns auf die Backend-Server-Infrastruktur und wir schlagen eine Methodik zur Identifizierung und Lokalisierung von IoT-Backend-Servern vor, welche von Cloud-Diensten und IoT-Herstellern betrieben wird. Wir analysierten Muster im IoT-Verkehr, der vom ISP beobachtet wird. Unsere Analyse gibt Aufschluss über die unterschiedlichen Strategien, wie IoT-Backend-Server betrieben und eingesetzt werden.

Die Notwendigkeit a-priori unbekannte netzwerkweite Anfragen an große Mengen von Netzwerk-Flow-Daten zu stellen, welche wir in in unseren Studien verwenden, motivierte uns zur Entwicklung von Flowyager. Dies ist ein auf bestehenden Netzwerkverkehrs-Tools aufbauendes System und es stützt sich auf die Zusammenfassung von Verkehrsflüssen, um (i) die Kosten für Archivierung und Transfer von Flow-Daten und (ii) die Antwortzeit von Anfragen zu reduzieren. Wir setzten einen Prototypen von Flowyager sowohl im IXP als auch im ISP ein.

Acknowledgments

During my Ph.D. journey, I had the privilege of working alongside some of the brightest people I've ever encountered. With this, I would like to express my deepest gratitude to Prof. Anja Feldmann, my advisor, without whom this journey would not have been possible. From 2016 when she gave me the chance to join her research group as a student worker, till today, she has always given me excellent advice and support—both in research and personal situations. We worked on many stressful deadlines that were sometimes late at night or early morning together; she showed me how to maintain my focus and get the job done. I've always looked up to her, admired her integrity, and appreciated the countless research and management insights she taught me. It was a privilege to work with you, Anja.

Georgios Smaragdakis, I'm extremely grateful to you for all the mentorship, advice, and support throughout the years. One should be very lucky to have a mentor like you. We had so many fun meetings that I lost count of them. Your dedication and attitude toward work are so encouraging that I enjoyed every moment of working with you. I learned so many things from you, from writing papers to formulating problem statements. Thank you for all the support that often went beyond just "work". As we promised, the next time we go for a coffee together, it will be on me.

My colleagues, many of them who are now my friends, have supported me at almost every turn of my studies. From jump-starting my career to giving technical, research, and personal advice and help. Apoorv Shukla, my friend, has always gone above and beyond to help me. He supervised my M.Sc. thesis and later supported me in becoming a colleague. We had memorable times, delicious Indian dinners, teas, and discussions in Berlin. I am grateful for all of that.

Thank you, Zubair Sediqi, an incredible friend and a dependable colleague, for all the good times we spent together. Time with you is always well-spent.

Dr. Srdjan Matic, thank you for always checking on me. I really enjoyed collaborating with you and Dr. Oliver Gasser. Mirko, we spent long hours organizing the courses. I have always admired your discipline and dependability. I would like to thank Lars, a talented researcher with an awesome sense of humor, has always reached out to help his colleagues; Florian Streibelt, Franziska Lichtblau, Ingmar Poese, Aniss Maghsoudlou and Seifeddin Fathali who never said No to my requests. Niklas Semmler, Mohammad, Sayed Ali, Cristi, Emilia, Danesh, and Bala, I wish you success in your endeavors.

I worked with talented students Arthur and Peter. Thank you both for all your hard work and efforts. A special thanks to our admin team and executive secretary, Rainer, Joerg, and Iris, for their excellent assistance and ensuring that infrastructure and administrative works run smoothly.

Finally, I would like to thank my family, who are the single most valuable thing in my life. My wife, Sheela, patiently supported me and our son, Said Ibrahim. With me being away or stressed with the deadlines, she suffered a lot, yet, she always had my back. My Mum and Dad, Zahra and Said Kamaludin are the two people I really don't know how to thank. I am eternally grateful for all their sacrifices. They nurtured me and supported our family at every turn so we could reach this point in our lives. My siblings, Dr. Said Jalal Saidi and Farida Said, have always inspired me. We supported each other through the good times as well as dark times.

This work was supported in part by the European Research Council (ERC) grant ResolutioNet (ERC-StG-679158).

Publications

Pre-published Papers

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

International Conferences

Said Jawad Saidi, Anna Maria Mandalari, Roman Kolcun, Hamed Haddadi, Daniel J Dubois, David Choffnes, Georgios Smaragdakis, and Anja Feldmann. "A haystack full of needles: Scalable detection of iot devices in the wild". In: *Proceedings of the ACM Internet Measurement Conference*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 87–100. ISBN: 9781450381383. DOI: 10.1145/3419394.3423650. URL: https://doi.org/10.1145/3419394.3423650

Said Jawad Saidi, Srdjan Matic, Oliver Gasser, Georgios Smaragdakis, and Anja Feldmann. "Deep Dive into the IoT Backend Ecosystem". In: *Proceedings of the 22nd ACM Internet Measurement Conference*. IMC '22. Nice, France: Association for Computing Machinery, 2022, pp. 488–503. ISBN: 9781450392594. DOI: 10.1145/3517745.3561431. URL: https://doi.org/10.1145/3517745.3561431

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. DOI: 10.1109/TNSM.2020.3034278

Said Jawad Saidi, Oliver Gasser, and Georgios Smaragdakis. "One Bad Apple Can Spoil Your IPv6 Privacy". In: *ACM Special Interest Group on Data Communications(SIGCOMM) Computer Communication Review* 52.2 (June 2022), pp. 10–19. ISSN: 0146-4833. DOI: 10.1145/3544912.3544915. URL: https://doi.org/10.1145/3544912.3544915

Workshops and Poster Sessions

Said Jawad Saidi, Damien Foucard, Georgios Smaragdakis, and Anja Feldmann. "Flowtree: Enabling Distributed Flow Summarization at Scale". In: *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*. SIGCOMM '18. Budapest, Hungary: Association for Computing Machinery, 2018, pp. 30–32. ISBN:

9781450359153. DOI: 10.1145/3234200.3234225. URL: https://doi.org/10.1145/3234200.3234225

Pre-published Papers: Not Part of This Thesis

I have contributed to the following peer-reviewed papers that have already been published but are not part ot this thesis.

Peer-reviewed Journals

Apoorv Shukla, Said Jawad Saidi, Stefan Schmid, Marco Canini, Thomas Zinner, and Anja Feldmann. "Towards Consistent SDNs: A Case for Network State Fuzzing". In: *IEEE Transactions on Network and Service Management* 17.2 (2019), pp. 668–681. DOI: 10.1109/TNSM.2019.2955790

Contents

1 Introduction

The introduction of the Internet of Things (IoT) has led users and businesses to deploy many sensors and smart Internet-connected devices in homes, factories, stores, cities, and remote locations. Estimates indicate that the number of deployed devices has already surpassed the human population, and it will reach 28 Billion by 2025 [7]. IoT devices that are installed at homes, provide a wide array of services, e.g., smart speakers, home appliances, and surveillance cameras, to name a few. Companies deploy them to gather insights and control their business and manufacturing processes. Typically, IoT devices, as clients, rely on a server infrastructure on the Internet to provide their services. Together with their backend servers, IoT devices form the IoT Ecosystem.

This ecosystem's health and secure operation are crucial to the continuity of IoT services and the security of the Internet. While the IoT ecosystem provides attractive services to home and industry users, they are often unaware of the security and privacy consequences associated with these services. On the client side, exploited IoT devices can (i) be used as an attack vector to infiltrate networks and exfiltrate private user information [8] (ii) be used as a weapon to launch large-scale coordinated attacks on other targets [9], (iii) become attack victim themselves, such that an adversary disables or limits the functionalities of an IoT service [10]. On the server side, a disruption stemming from a successful denial of service or a compromising attack on the IoT backend server infrastructure lets an adversary disable or control a large number of IoT devices [11].

As the IoT ecosystem grows [7], researchers and network operators are curious to have a better understanding of this ecosystem. They are interested to know the population and type of deployed IoT devices and their interactions with backend servers. Moreover, users are interested to know the security or privacy consequences of operating such devices. IoT companies want to understand the security risks threatening their services and infrastructure [12]. A characterization of the IoT ecosystem *at scale* can yield insights about its state, evolution, and security and privacy risks. These insights can serve the curiosity of Internet researchers and network operators, and subsequently, can be used to mitigate threats involving the IoT ecosystem.

Understanding and characterizing the IoT ecosystem is challenging though, partly due to its rapid growth, evolution, and heterogeneity. As mentioned above, deployed IoT devices are increasing, and so are the infrastructure to support them. New IoT companies are rolling out IoT devices, services, and applications into the market, and many cease to exist [13]. These frequent changes add to the heterogeneity of deployed IoT devices.

IoT devices have diverse traffic patterns. They use a wide range of protocols to communicate with a diverse set of backend servers. For example, they may rely on general-purpose application layer protocols such as Hyper Text Transfer Protocol (HTTP) or use specialized IoT-related protocols, such as Message Queueing Telemetry Transport (MQTT). Some devices rely on a few backend servers, while others contact many destinations. The frequency of communication and the volume of exchanged data between IoT devices and their servers are also diverse. As an example, watching videos using Smart-TVs requires downloading large data volumes, while home cameras are typically upload-intensive (sending video feeds).

On the server side, companies rely on different strategies to set up their backend server infrastructure to manage, ingest data, and monitor their fleet of deployed IoT devices. They set up their IoT backend servers using one or a combination of dedicated infrastructure, cloud services, or shared infrastructure such as Content Delivery Networks (CDNs). The market demand for IoT backend servers has resulted in the establishment of specialized IoT backend provider companies that sell specialized IoT backend server infrastructure and related services. Popular public cloud providers such as Amazon, Google, and Microsoft [14–16] have also entered this market and added the IoT backend server *as-a-service* to their product portfolio.

So far, solutions often use active measurement techniques to analyze and characterize the IoT ecosystem on the Internet. A wealth of research and solutions scan the IPv4 Internet to detect and enumerate the IPv4 addresses hosting IoT devices or servers [17–26]. They analyze banners or look for the presence of IoT-related protocols to infer the presence of an IoT device or server. If possible, these solutions also provide information on the detected entity's type, make, and model. For example, an IPv4 address hosts an IoT device which is an IP camera from manufacturer X. Furthermore, they often augment these datasets with metadata such as Autonomous System Number (ASN), banner, geo-location, and security analysis. Kumar *et al.* [27] deployed agents inside the premise of end-users and scanned the home network for deployed IoT devices.

Existing active measurement solutions provide invaluable insights about millions of deployed IoT devices and servers. However, their methodologies (i) only report the behavior of targets in response to probe packets, not the traffic as observed through passive monitoring [28], (ii) are intrusive to the end-users, and raise privacy concerns if we deploy agents at the end-users' premises, (iii) are challenging to identify devices that reside behind a Network Address Translation (NAT), (iv) may miss many IoT backend servers due to protocol handshake failures and services running on unexpected port numbers [20], and (v) cover almost exclusively IPv4 Internet and may not easily scale to cover IPv6 Internet (due to the vast number of IPv6 addresses).

A major body of work use passively collected network traffic data to gain insights into the activity of IoT devices. Such studies complement the active measurement approaches with insights that otherwise are not readily observable. For example, with an Internet Service Provider (ISP) as a vantage point, researchers can potentially observe the traffic patterns of thousands of devices and servers.

To this end, passive approaches start by detecting the IoT devices in network packet captures [29] or flow captures such as NetFlow [30] and IPFIX [31]. They develop fingerprints (signatures) of IoT devices by extracting unique features of devices' network traffic. Alternatively, they train a machine learning model to learn these fingerprints [32–35]. They infer the presence of an IoT device by finding these signatures in network traffic.

Deploying passive IoT device detection approaches *at scale* in networks such as ISPs, and IXPs is challenging. For example, solutions that require the analysis of full packet captures in an ISP with millions of subscribers and 10TB/s peak traffic require substantial resources and investment. Thus, although several passive approaches deployed their solutions in smaller environments, e.g., test-beds [33, 36], home networks, campus and enterprise networks [35, 37], only a few studies have attempted to deploy their solutions at *scale* and report *statistics* on the number of detected devices. [1, 34, 38, 39]. Note, these studies mostly took place either concurrent with or after our studies.

We looked into the related research on the characterization of IoT backend servers as observed in the passively collected data. Despite the critical role of IoT backend servers in the overall health and security of the IoT ecosystem [11, 40], compared with the IoT device detection research, it has received significantly less attention. To our surprise, for the activity of IoT backend servers, we found only one study by Mazhar *et al.* [28] that briefly discusses the IoT backend servers at a smaller scale. Their analysis revealed that while the IoT space is fragmented, few popular cloud and DNS services act as a central hub for the majority of the devices and their data.

Major network operators, including ISPs and Internet eXchange Points (IXPs), are taking steps to combat attacks on and from the IoT ecosystem. As most IoT device operators are subscribers of ISPs, these providers, with millions of subscribers, are often the first or last line of defense against attack traffic before it reaches or after it leaves the customer networks. Hence, as explored by recent works [41, 42], ISPs have a significant potential (i) to observe the state of the IoT ecosystem at scale (ii) to detect IoT-related security incidents (iii) help their customers to mitigate attacks involving the IoT ecosystem.

To this end, large network operators are developing methodologies and systems to detect and handle security and privacy threats involving IoT ecosystem [41, 42]. One natural first step toward developing these methodologies is detecting and enumerating the IoT devices and their backend servers that communicate through the provider's network. Once detected, providers will be able to measure their traffic patterns and obtain a better understanding of this ecosystem. Specifically, ISPs are interested in estimating the population of different IoT devices, their destinations, i.e., servers, protocols, and the characteristics of their exchanged traffic. These insights aid ISPs in identifying and measuring the security and privacy risks common among their IoT subscriber lines. In summary, detecting IoT devices and servers, studying their traffic patterns, and identifying and measuring security privacy and risks using large vantage points such as ISPs and IXPs, allow us to characterize the IoT ecosystem at scale.

However, to use ISPs and IXPs as vantage points, we have to deal with the poor availability and low granularity of data sources. The available data sources are often in the form of large volumes of passively collected, aggregated, and sampled data, e.g., NetFlow [30] and IPFIX [31], which providers routinely collect for their operational purposes. Thus, for the characterization of the IoT ecosystem, we need methodologies that handle large volumes of sparsely-sampled data and do not rely on packet payloads.

Hence, three factors motivate this dissertation: (i) The IoT ecosystem is rapidly growing and evolving. (ii) The threats involving the IoT ecosystem are increasing. (iii) Large network providers need methodologies to handle these threats.

1.1 Dissertation Goal

In this dissertation, our overarching research goal is the following question: How can we, using the data-driven methodologies, characterize the IoT ecosystem at scale? To find an answer to this research question, we strive to answer the following subquestions:

(i) How to detect IoT devices using sparsely-sampled flow capture data from a large vantage point such as an ISP or IXP?
(ii) How to detect IoT backend server infrastructure on the Internet and characterize their deployments?
(iii) How to characterize the traffic patterns of IoT devices and backend servers as observed in an ISP?
(iv) How the deployment of IoT devices at homes can affect users' privacy?

Since our efforts require exploring large volumes of flow capture data and issuing apriori unknown queries, we also considered How do we improve response time while querying large volumes of network flow capture data?

Our methodologies aid researchers and network operators in obtaining a more detailed view and understanding of the IoT ecosystem. Our insights can inform the decision process and policy-making of Internet-governing organizations and network operators when dealing with the IoT ecosystem and its threats.

1.2 Contributions

As we intended to perform a data-driven study of the IoT ecosystem at scale, we collaborated with a large European ISP and a large European IXP to have them as our vantage points. We start with one of the main components of the IoT ecosystem: IoT devices. We developed methodologies to detect and enumerate IoT devices in the wild. Then, we performed a case study on how the deployment of millions of IoT devices can affect the privacy of ISP's subscriber lines. We identified a privacy leakage common among millions of IPv6 subscribers, showing that IoT devices are the major contributors to this privacy leakage. Finally, we turned our attention to IoT backend servers by performing one of the first studies on the IoT backend providers. We developed methodologies to detect the Internet-facing infrastructure of top IoT backend providers. We analyzed their traffic patterns, security incidents, and outages.

Our methodologies rely on large volumes of sparsely sampled and passively collected flow captures, such as Netflow and IPFIX, from our vantage points. Hence, we also designed and implemented a system called Flowyager that allows network operators to store, index, and query flow-capture data interactively.

Our four main contributions are as follows:

1. **Scalable Detection of IoT Devices in the Wild**
   We developed a methodology for identifying IoT devices by focusing on their back-end infrastructure's domains and IP addresses. To this end, we derived distinct signatures in terms of IP/domain/port destinations to recognize IoT devices. With our signatures, we were able to recognize the presence of devices from 31 out of 40 manufacturers in our testbed. Moreover, we showed that it is possible to detect the presence of IoT devices at subscriber lines using sparsely sampled flow captures from a large residential ISP and a major IXP, even if the device is idle, i.e., not in active use. Specifically, we were able to recognize that 20% of 15 million subscriber lines used at least one of the 56 different IoT products in our testbed. Finally, we highlighted that our technique scales, is accurate, and can identify millions of IoT devices within minutes in a non-intrusive way from passive, sampled data. In the case of the ISP, we were able to detect the presence of devices from 72% of our target manufacturers within 1 hour, sometimes minutes.
2. **IoT-driven Privacy Leakage of IPv6 Users**
   We performed a case study at a large European ISP on how the deployment of IoT devices can affect the privacy of subscriber lines. We identified a privacy leakage related to the usage of legacy IPv6 addressing mechanism EUI-64. We showed that the existence of even a single device without privacy extensions in an end-user prefix could defeat the ISP-deployed prefix rotation and IPv6 privacy extensions adopted by hardware vendors to preserve user privacy. We measured the impact of this privacy leakage in terms of the number of affected subscriber lines. Our analysis found that around 19% of end-user prefixes host at least one device that does not use IPv6 privacy extensions. We also showed that a popular content provider, application, or service contacted by a device not using privacy extensions, can track the user and other contacting devices across rotating prefixes. We performed a root cause analysis and revealed that IoT devices contribute the most to this privacy leakage. In most cases, a single device without privacy extensions was responsible for the privacy leakage. Unfortunately, these devices have been manufactured by market leaders. Thus, it would have been possible to prevent this privacy leakage if these

manufacturers had adopted best common practices, i.e., IPv6 privacy extensions. We, finally, provided recommendations to remedy the privacy violation.

3. **A Study on the Infrastructure of IoT Backend Providers**
We developed a methodology to infer the network and location of major IoT backend providers. Our methodology relies on a fusion of information from public documentation, passive DNS, and active measurements. We analyzed the IoT backend servers with regard to deployment, operation, and dependencies. While most of the popular IoT backend providers have footprints covering multiple locations and countries, our analysis showed that some operate only in one country or rely on infrastructure from other IoT backend providers. We found that it is not unusual for IoT protocols, e.g., MQTT, to use non-standard ports or reuse Web ports. The latter makes the identification of IoT backend infrastructure as well as IoT traffic challenging using traditional methods—our proposed methodology resolves this issue. Using passive data from a large European ISP, we examined the IoT traffic patterns of multiple providers at scale. We noticed that a significant fraction of IoT Traffic —more than 35%— is exchanged with IoT backend servers outside Europe, which raises performance and regulatory concerns. Our traffic analysis highlighted that the IoT population and activity per application differ vastly. While some applications behave more like the typical user-generated traffic, i.e., diurnal patterns, peak evening hours, and were downstream-heavy, this was not the case for all IoT applications.
4. **Interactive Exploration of Network-Wide Flow Data**
We designed, deployed, and evaluated Flowyager, a system that is built on the existing voluminous network captures and enables interactive data exploration. We showed that with Flowyager, the query response time for network-wide queries can be reduced from hours or minutes to seconds. We proposed 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 share our experience of rolling out Flowyager in different operational environments, namely a large IXP and a tier-1 ISP, and showcase how to tackle various network management tasks. We made Flowyager and its code available for non-commercial use.

1.3 Collaborations and Pre-published Work

Chapter 5: IoT Devices: Detection in the Wild

This chapter is based on a paper published at IMC 2020 conference, in collaboration with the authors in [1]. Specifically, the author's main contributions to this paper are as follows: (i) joint development of the methodology to generate IoT detection rules, (ii) analysis of ISP and Home vantage point data, and (iii) plotting and joint narrative conception and writing.

Chapter 6: IoT Devices: A Case Study on Leaking Users' Privacy

This chapter presents a paper published at CCR 2022 with the collaboration of authors listed in [4]. Our main contributions to this paper are as follows: (i) joint conception of the scenario for privacy violation, (ii) development of the methodology to classify devices using EUI-64 addressing, (iii) analysis of passive ISP data, and (iv) plotting, and joint narrative conception.

Chapter 7: IoT Backend Servers: A Deep Dive into Backend Providers

Chapter 7 presents a paper that is published in the proceedings of the IMC 2022 conference. This paper is the result of the collaboration with the authors noted in [2]. In particular, the following are the author's main contributions: (i) joint development of the methodology to extract IoT-related domains and build regular expressions, (ii) analysis of ISP dataset, and (iii) visualization, joint narrative conception, and writing.

Chapter 4: Flowyager: Exploring Network-Wide Flow Capture Data

Chapter 4 presents a paper that with the collaboration of authors listed in [3] is published in the IEEE TNSM journal. The author's main contributions to this paper are as follows: (i) joint development and implementation of Flowtree data structure, (ii) evaluation of the Flowtree data structure, (iii) joint design of Flowyager system, (iv) implementation and evaluation of FlowDB and FlowAGG modules, (v) joint data visualization, plotting, narrative conception, and writing.

1.4 Thesis Structure

The structure of this dissertation is organized as follows:

Chapter 2: Background

This chapter lays the foundation with the necessary concepts to better understand the rest of the thesis. It provides an overview of the structure of the Internet, the role of ISPs and IXPs in the Internet, and network flow capturing utilities.

Chapter 3: Setup for IoT Studies

This chapter describes the concepts, terminologies, and experiment settings shared by Chapters 5- 7. We start with an introduction to the concept of the Internet of Things (IoT). Next, we introduce the client-server landscape of the IoT ecosystem and the important protocols used for communication between IoT devices and backend servers. We provide an overview of existing solutions for detecting IoT devices in networks by explaining their data sources and methodologies. We further highlight the need for scalable methodologies that detect IoT devices using passively-collected, sparsely-sampled network flow data. We introduce the role of IoT backend servers in the IoT ecosystem in general and highlight the role of IoT backend provider companies.

Finally, this chapter describes the settings of two large networks we used for data collection in our studies: a large Tier-1 European ISP and an international IXP. We explain the data sets collected at each network, and more importantly, we discuss the ethical considerations we undertook for our studies.

Chapter 4: Flowyager: Exploring Network-Wide Flow Capture Data

In this chapter, we investigate the problem of improving the response time in a priori unknown network-wide queries in environments such as ISPs and IXPs. Many network operations require answering network-wide flow queries in seconds, ranging from attack investigation and management to traffic management. ISPs and IXPs collect network flow records such as Netflow and IPFIX for operational purposes. Although flow records are collected at each router, using available traffic capture utilities, querying the resulting datasets from hundreds of routers remains a significant challenge due to the sheer traffic volume and distributed nature of flow records. Hence, we present Flowyager, a system that is built on top of existing traffic capture utilities. Flowyager generates and analyzes tree data structures that we call Flowtrees, which are succinct summaries of the raw flow data made 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 a large IXP and an ISP, we showcase its capabilities for networks with hundreds of router interfaces.

Chapter 5: Detection of IoT Devices in the Wild

In Chapter 5, we focus on the IoT device component of the IoT ecosystem. We developed a methodology to detect IoT devices using passively collected and sparsely sampled flow data from a large European ISP and a European IXP. We built on the insight that IoT devices typically communicate with their backend server infrastructure to provide their services. Hence, they generate traffic patterns, i.e., a set of domains, IP addresses, and port numbers that can be used to identify subscriber lines with IoT devices.

We started by investigating the visibility of the traffic from a single device on a single subscriber line in ISP data. Then, we investigated at what granularity we can detect IoT devices and how fast we can detect them in the sampled datasets. We performed a large-scale study in a large European ISP and IXP to identify the number of subscriber lines hosting the studied IoT devices. Finally, we make a case for ISPs to detect IoT devices by discussing their potential security benefits.

Chapter 6: IoT Devices: A Case Study on Leaking Users' Privacy

In Chapter 6, we investigate how the deployment of IoT devices can affect the privacy of millions of subscribers of a major European ISP. We demonstrate that using legacy IPv6 address assignment mechanisms such as IPv6 EUI64 (Extended Unique Identifier) can make subscriber lines prone to tracking. We show that ISPs and many device manufacturers use best common practices for privacy, namely prefix rotation and IPv6 privacy extensions. Yet, despite these efforts, a single device with

IPv6 EUI64 on the home network can spoil the privacy of potentially all IPv6-enabled devices and, eventually, end-users' privacy across these devices.

By analyzing passive data from a large ISP, we find that around 19% of end-users' privacy can be at risk. Our results show that IoT devices contribute the most to this privacy leakage and, to a lesser extent, personal computers and mobile devices. To our surprise, some of the most popular IoT manufacturers have not yet adopted privacy extensions that could otherwise mitigate this privacy risk.

Chapter 7: IoT Backend Servers: A Deep Dive into Backend Providers

In this chapter, we turn our attention to the backend server component of the IoT ecosystem. In particular, we conducted one of the first studies on IoT backend server providers. IoT backend providers are companies that offer software and server infrastructure to manage, control, and monitor IoT devices and ingest their data.

Given the limited resources on the devices and the increasingly rich and resource-intensive applications of IoT devices, we underline the importance of offloading some of these functionalities to IoT backend servers in the 'cloud.' Recently, the big technology giants, such as Amazon [14], Google [15], and Microsoft [16] started to offer IoT backend solutions as a service. Such companies are IoT backend providers and enable third-party IoT application providers to scale up and deliver their solutions to potentially billions of IoT devices deployed around the globe.

Despite the critical role that these IoT backend providers play in the operation and security of IoT applications, little is known about their locations, strategies, and volume share. We developed new methods to identify their footprints and gain insights into their modus operandi. This chapter describes our methodology to detect and infer the location of the Internet-facing infrastructure of these providers. We characterize their traffic patterns as observed in a large European ISP. Finally, we conducted a case study to measure the impact of an outage in a large public cloud provider on the services of IoT backend providers.

2 Background

Internet is woven into various parts of modern society's fabric. Many critical systems, such as financial systems, telecommunications, transportation control, and military systems – to name a few – rely on the Internet to provide their services. Some countries have even codified Internet access as a human right [43] into their laws.

The Internet was initially designed as a network primarily used for military purposes. Since its inception, the Internet, as we have it today, has experienced significant growth and has evolved into a complex global network connecting billions of users and devices worldwide. To this extent, although built by humans, the Internet has become a phenomenon that we, humans, no longer know its exact topology. There is much more unknown about the Internet than what is known about it. Hence, researchers worldwide are actively using various measurement techniques to characterize and understand the current state of the Internet.

However, there are certain structures and guiding principles that allow the Internet to still work. Recent studies have shown that it can even handle the substantial amount of traffic that was suddenly added due to the lockdowns imposed during the COVID-19 global pandemic [44]. In this chapter, we describe the structure of the Internet, the role of ISPs and IXPs, and their monitoring technologies.

2.1 Internet Structure

This section provides an overview of how the Internet's structure has evolved over the past decades. The terminology and figures are based on [45].

2.1.1 Traditional View

Traditionally, the structure of the Internet, as described in classic computer networking text books [45], is modeled into three layers structure as shown in Figure 2.1. The end-users who want to access the Internet pay an Internet Service Provider(ISP) to obtain connectivity. These ISPs are called access/Tier-3 ISPs and are at the bottom layer of the model(See Figure 2.1). Such ISPs provide connectivity services to a limited geographical region, typically at the level of cities or sometimes a country.

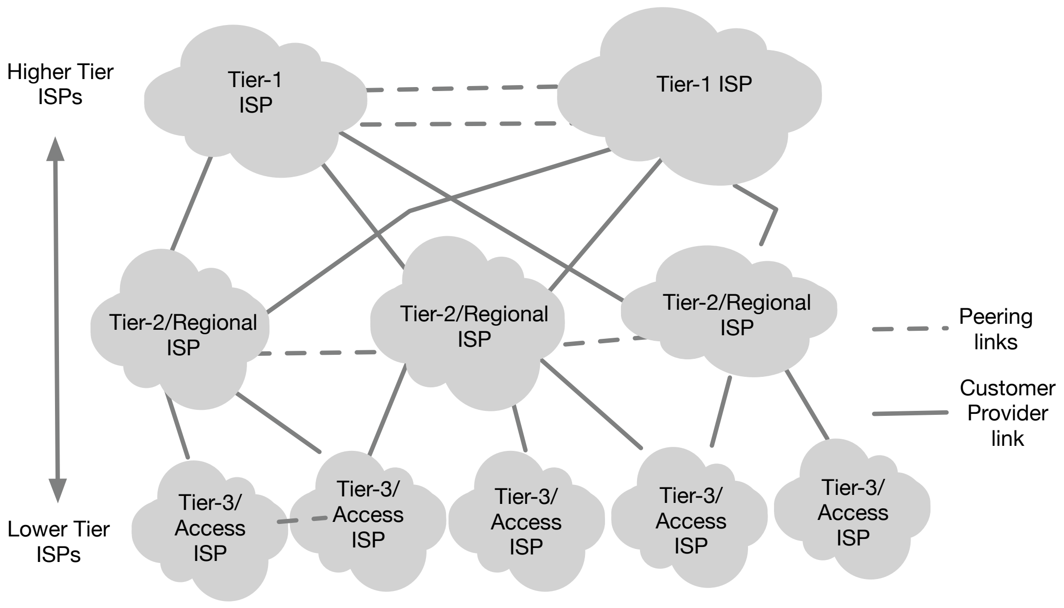


Figure 2.1: *Traditional Model of Internet Structure. Figure based on [45]*

When a user fetches content, e.g., visits a website or watches a movie, in the background, their device sends a request to a computer called a server, which typically sits in a data center somewhere in the world. In this scenario, the access ISP connecting the user is called *eyeball ISP*.

Upper in the hierarchy, access ISPs pay larger Regional ISPs, which serve larger geographical regions such as multiple countries, to carry their traffic. When an ISP pays another to forward its traffic, we say there is a Customer-Provider relationship between them. Regional ISPs pay the Tier-1 ISPs to forward their traffic. However, sometimes, two ISPs establish *peering aggreements* by setting up direct links and forwarding each other's traffic free of charge.

At the top layer of the Internet are Tier-1 ISPs. These are large carriers whose infrastructure span multiple continents and even around the globe. They are heavily interconnected and exchange traffic among each other free of charge.

2.1.2 Modern View

The strictly hierarchical model demonstrated in Figure 2.1 no longer describes the structure of the Internet. During the past decades, the Internet has undergone substantial transformation to a more *flat Internet* [46], which we illustrate in Figure 2.2.

As mentioned earlier, two ISPs may establish direct links and exchange traffic free of charge; such a business relationship is called *peering.* This idea became popular, and the community decided to scale it by having multiple networks gather at a shared data center facility to establish *peering* links. This would allow more networks to benefit from the advantages of peering relationships and reduce the cost of setting up

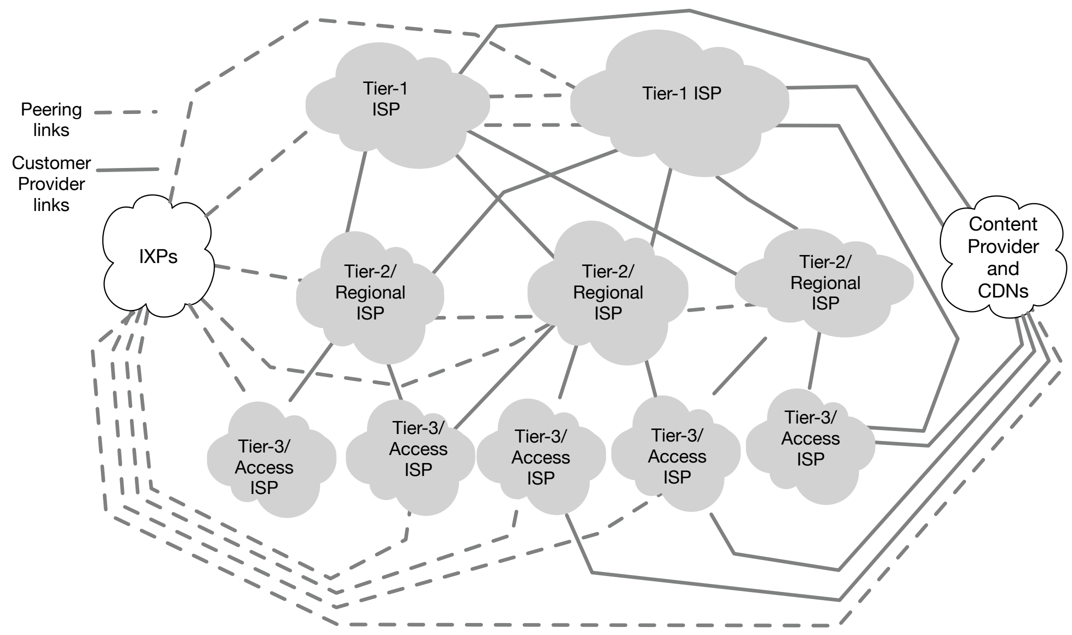


Figure 2.2: *Flat Internet Structure: Rise of IXPs and Content Providers. Figure based on [45]*

interconnections. As such, Internet Exchange Points(IXP) came into existence. We further elaborate on the anatomy of an IXP in Section 2.1.3.

Large content provider companies, e.g., Google and Facebook, have built several data centers and deployed extensive network infrastructure worldwide. They are establishing direct connectivity to the access ISPs, bypassing the higher Tier ISPs, thus saving costs and having more control over how their traffic is delivered to the end-users[45].

Similar to content providers, Content Distribution Networks (CDNs) and cloud services, e.g., Amazon AWS [47] and Microsoft Azure [48] have also established data centers in multiple locations and laid their private network infrastructure spanning around the globe. A prominent example of CDNs is Akamai [49], which places servers closer to users, thus seeking connectivity to more access ISPs. The interconnection of content providers, CDNs, and cloud providers with the access ISPs, and the rise of IXPs, have made the Internet's structure complex and flat.

2.1.3 Internet Exchange Points

In this thesis, we collaborate with an IXP as one of our vantage points to collect our datasets. IXPs provide physical infrastructure for multiple networks to establish peering links in a shared facility. Their main business model is based on setting up a layer two switching fabric and selling its ports to the members. Networks that want membership in IXP shall install their border routers in the IXP facility and

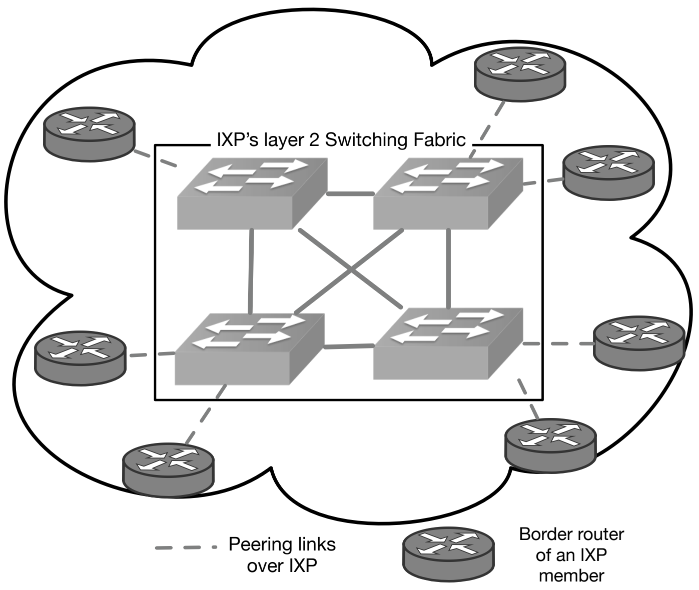

IXP's layer 2 Switching Fabric
Peering links over IXP
Border router of an IXP member

Figure 2.3: *Architecture of an Internet Exchange Point(IXP): IXP members connect their border routers to the switching fabric of IXP and establish peering links.*

connect their routers to the switching fabric. Figure 2.3 illustrates an overview of the architecture of an IXP.

Note that the IXP members are not necessarily all ISPs; Enterprise, Academic, and Government networks may also become members. Nevertheless, each member has a different *peering policy*, mainly driven by its business objectives. For example, members with an open peering policy typically establish peering links with as many networks as possible. On the other hand, members with a selective peering policy establish peering links if the link fits their business objectives. Large Tier-1 providers typically have a selective peering policy and establish peering links with equally large providers.

2.2 Network Flow Capturing

Network operators are interested in analyzing their traffic to gain insights into networks' health, security, and status over multiple time periods. For example, they may want to know the top 10 applications/port numbers contacted by their users in the past week. Another example is the queries for detecting and investigating an attack, e.g., the source IP addresses generating attack traffic or the destination IP addresses (attack targets).

Network operators have to deal with large volumes of network traffic to answer such queries. Consider a network with thousands of routers distributed at different geographical locations that carry several Terabytes of traffic per second. In such an environment, one may be tempted to capture the network traffic, then transfer them to a central location for storage and processing. However, given the sheer size of the network traffic, such approaches are prohibitively expensive.

Therefore, operators typically rely on network flow capturing utilities, such as NetFlow, IPFIX, and sFlow, to keep track of *Network Flows* in their network devices. A network flow describes of series of packets with the same five tuples in their headers. These five tuples are source IP(src IP), destination IP(dst IP), source port number(src port), destination port number(dst port), and transport layer protocol(proto). Moreover, a network operator may configure the capturing utility to collect other metadata such as the number of packets/bytes in the flow, start and end timestamps, src/dst MAC addresses, device ID, interface ID, the direction of flow and so forth.

To this end, the flow capturing utility aggregates the packets into flows and increments their respective counters. Finally, it exports the *Flow Records* to a flow collector when one or more of the following events happen: (i) one or more flows complete, (ii) the device has to release resources, (iii) a maximum time out for keeping track of a flow is reached. A flow collector saves the flow records to a disk for further processing. Thus, by keeping only the header information of the packets, flow capturing utilities reduce the storage and bandwidth requirements.

Still, capturing and generating statistics, e.g., packet and byte counts, for each flow may require substantial resources of a networking device –especially in large networks with terabytes of traffic per second in their peak time. Moreover, the volume of exported flows may also be quite large. To further reduce resource usage, the network operators configure the flow capturing utilities to perform *sampling*. They configure the capturing utility with the sample rate of $\mathbf{1/n}$. The sampling rate instructs the capturing utility to randomly or deterministically select one packet out of $\mathbf{n}$ packets for processing. With the 1:1 sampling rate, all packets will be effectively selected for processing. On the other hand, with the lower sampling rates, the amount of resources for capturing and storing the flow records and the accuracy of the statistics calculated from these records decreases. Moreover, the odds of missing some of the flows with sparsely sampled flow data also increase.

In the following subsections, we briefly introduce two prominent flow capturing utilities, which are also used in our studies.

2.2.1 NetFlow

NetFlow is a flow capturing utility introduced in Cisco networking devices [50]. Network admins enable it and select a sampling rate on network device interfaces. Then, the device selects one out of $\mathbf{n}$ consecutive packets for NetFlow processing. The NetFlow process uses the previously-mentioned five-tuple fields from the packet to create a flow record. As further packets of a flow are sampled, Netflow updates the statistics of that flow, e.g., updates the packet/byte count. NetFlow can keep track of the state of a flow; if it detects a flow termination, e.g., by detecting a TCP-FIN, TCP-RST flag, or expiration of a timeout counter, it exports the flow record to a flow collector. NetFlow uses port 2055 of UDP to export the flow records. Although Cisco originally introduced NetFlow, many vendors have added support for this protocol to their devices. The latest version of Netflow is v9 and defines 79 field types.

2.2.2 IPFIX

The success of the NetFlow protocol persuaded the Internet community to gather and create a standardized flow capture protocol. The Internet Engineering Task Force(IETF), specified the IP Flow Information Export(IPFIX) protocol in RFC 5101 and 7011 [31, 51]. IPFIX is an open standard protocol based on NetFlow. It provides backward compatibility to the NetFlow v9 by defining the same data fields and types. However, IPFIX supports even more data fields, and so far, 491 data fields have been registered by IANA [52].

IPFIX works similarly to NetFlow and exports the flow records to flow collectors. Device administrators can configure the IPFIX process to include different data fields in their exported flow records. A combination of data fields is called **template**. Apart from the flow records, an IPFIX exporter periodically sends its templates to the collectors. The default port number for exporting IPFIX packets is 4739.

2.3 Exploring Network-wide Flow Capture Data

Network operators have to continuously keep track of the activity in their networks 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 flow-level or packet-level captures from routers within their network [53]. They also need systems to process these captures and do important network-management tasks. We provide a summary of tasks and how previous work tackled them in Table 2.1.

In the rest of this chapter, we describe (i) the problem of issuing a priori unknown queries to explore network capture data, (ii) an overview of related work for exploring network-wide flow data.

2.3.1 A Priori Unkown Queries

Recently, query-driven solutions, e.g., Sonata [59], Stroboscope [70], and Marple [60], 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)

Application	Related Work
Aggregated flow statistics (range queries over IP/ports/time/location)	[54–60]
Counting traffic	[54, 57–59, 61–66] [67–74]
Traffic matrix	[63, 67]
DDoS diagnosis	[59, 61, 63, 67, 72, 75–77]
Super-spreaders Detection	[59, 61, 72]
top-K number of flows	[62, 73, 74, 78]
Flows above threshold T (Heavy Hitters)	[56–59, 61, 62, 69, 71, 72, 79, 80]
Heavy Changers Detection	[61, 71, 72, 79–81]
Blackhole Detection	[63, 82–84]
Port-based / 4/5-tuple queries	[54, 59, 61–63, 68, 69, 71]

Table 2.1: *Typical network queries and systems to tackle them. Currently, no system addresses all of them.*

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 required queries are not known in advance. Network engineers often have to try different queries to locate the source and type of problem interactively. Thus, compiling such queries into telemetry programs takes a prohibitively large time . 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 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 [50, 85], sFlow [86], IPFIX [31], or libpcap [29] 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 2.1). 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 [87], 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.

2.3.2 Related Work

There are existing network analytics systems such as [88, 89] that 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) [90]. Fines are steep, namely up to 4% of worldwide turnover or 20 million Euros, whichever is higher. In Chapter 4, we present our system Flowyager to tackle this issue.

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 [70], network troubleshooting using mirroring [91, 92], analysis of in-network packet traces [63, 93], as well as monitoring links on-demand as shown by Gigascope [54], pruning-based solutions such as Cheetah [94] or other SDN-based monitoring, such as [95] or PRECISION [96]. 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, from more classic approaches such as A-GAP [74] to the numerous modern solutions, such as Sonata [59], FlowBlaze [97] or Poseidon [98], 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 [99]. 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 [60] 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 [87] and Hadoop [100], or column-based databases, e.g., ClickHouse [101]. This has scalability issues. Thus, recently proposed big data analytic systems—see [102–106] as well as [107] 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. Such big data analytics systems are *flexible* w.r.t. the queries supported. Yet, 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*. We designed our system Flowyager in Chapter 4 to address these shortcomings.

Data summaries–Heavy Hitters: Previous work on computing network summaries has focused on how to efficiently compute heavy hitters (HH) [85, 108–110] and hierarchical heavy hitters (HHH) [57, 111, 112] 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 2.1), 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: popularities increase overall as more data comes in. Consequently, the threshold to be considered a heavy hitter should be raised. In contrast, some HHH data structures, e.g., [57] 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., [71, 113, 114] as well as systems that utilize sketches for network monitoring and debugging [61, 72, 115–117]. The capabilities of sketches include counting, top-K, HH, and 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.

2.4 Chapter Summary

This chapter briefly overviewed the Internet's structure and the role of ISPs and IXPs. We introduced two popular network flow capturing utilities: NetFlow and IPFIX. We underlined the importance of exploring the network flow captures to gain insights into the health and security of networks. We also introduced the problem of issuing a priori unknown queries to achieve important network management tasks and the extent the existing solutions address this problem.

3 Setup for IoT Studies

The previous chapter explained the necessary background to understand this thesis. This chapter lays the groundwork for our IoT studies by clarifying the essential terms such as IoT, IoT devices, IoT backend servers, IoT ecosystem, and IoT device detection. We discuss the related work in IoT device detection and characterization of the IoT backend servers. Several popular IoT protocols are also covered. We provide the details of our collaboration with a European ISP and a large IXP and the ethical considerations we took while handling their datasets.

3.1 Internet of Things Ecosystem: Terminologies

What almost everyone agrees about IoT is that there is no universally agreed-upon definition of IoT. Despite the increasing role and deployment of the Internet of Things (IoT), academia, industry, media, and organizations have provided various definitions for the term “IoT”, partly due to their own perspectives on the IoT and the ever-evolving nature of the IoT ecosystem. For example, International Telecommunication Union, a United Nations agency, defines IoT as “a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies.” [118] Amazon AWS, a major cloud computing company, defines IoT with respect to its relationship to the cloud: “[...] IoT refers to the collective network of connected devices and the technology that facilitates communication between devices and the cloud, as well as between the devices themselves.” [119]. Enumerating and analyzing all of the definitions provided for the IoT is beyond the scope of this thesis. Indeed, there have been attempts to bring various stakeholders together and provide a unified definition of IoT; one notable example is the “Define IoT” initiative by IEEE [120].

Nevertheless, the invariant part of these definitions is that IoT is a collection of network-connected objects. In this thesis, we require these objects to be connected to the *Internet*. As we perform a data-driven analysis of the IoT ecosystem in collaboration with ISP and IXP vantage points, non-Internet-connected objects will not be visible in our datasets. Hence, the objects in private networks without Internet connectivity and those which exclusively communicate over non-IP networks(for example, the devices that only use Short Message Service(SMS) for their communication) are beyond the scope of this thesis.

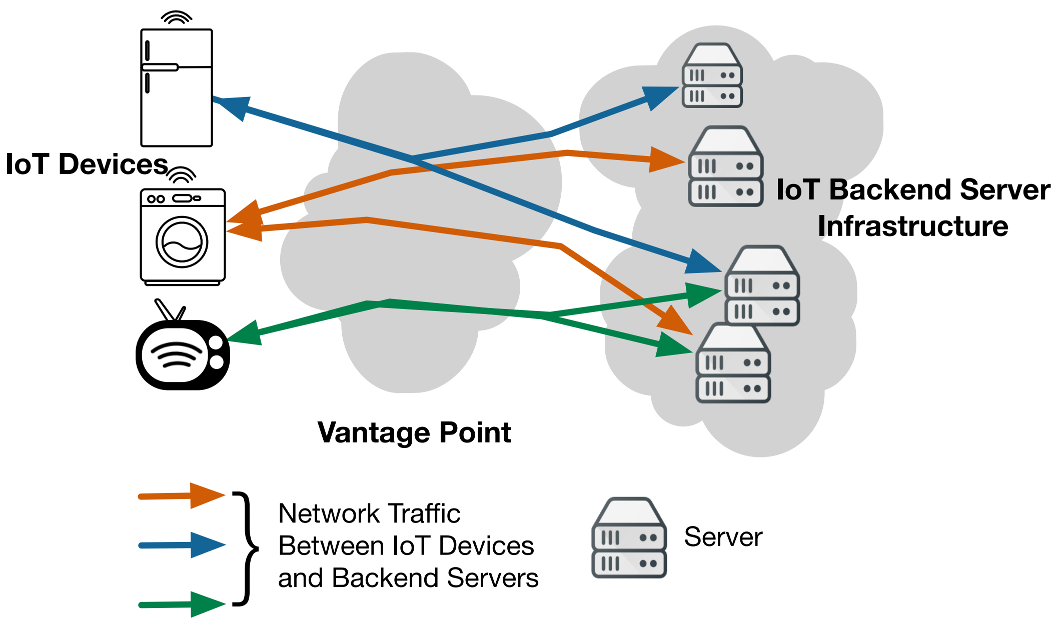

IoT Devices
IoT Backend Server Infrastructure
Vantage Point
Network Traffic Between IoT Devices and Backend Servers
Server

Figure 3.1: *IoT Ecosystem.*

See Figure 3.1, we envision a client-server communication model where a client relies on services offered by a server; they exchange traffic, and we observe their traffic from our vantage points. In this model, the network-connected objects that act as clients are **IoT devices**, and those who act as servers are **IoT backend servers**. To this end, the collection of IoT devices and their backend servers form the **IoT ecosystem**.

IoT devices may include sensors, controllers, actuators, and household appliances [121]. However, with the advancement of technology, manufacturers pack more processing power and functionalities into a single device. Indeed, we can install new applications on some devices and extend their functionalities. Thus, as first introduced by Mazhar *et al.* [28], we can devise a spectrum of devices based on their supported functionalities. See Figure 3.2 for this spectrum. On the left end of this spectrum, we see simple IoT devices with a single purpose and limited functionalities. These are sensors, actuators, smart light bulbs, and home appliances.

As we move to the right end of the spectrum, the number of functionalities and the processing power of the devices increase. On the right-most end, we find general-purpose devices such as laptops, personal computers, and smart mobile phones. In the middle, devices are less sophisticated than general-purpose ones but can have more functionalities than the left end of the spectrum. These devices, like Smart TVs and Smart Speakers, allow users to install a limited set of new applications and extend their basic functionalities. In this thesis, we exclude the right-most part of this spectrum as they are not generally considered IoT devices [28].

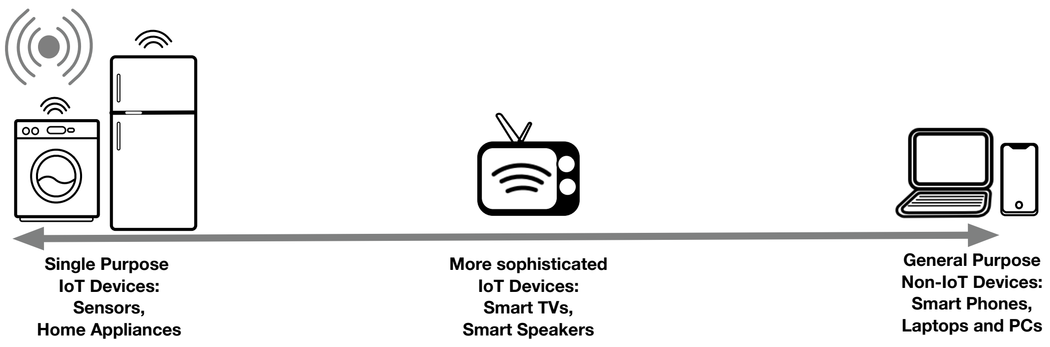


Figure 3.2: *Spectrum of IoT devices to Non-IoT devices. Figure based on Mazhar*et al. *[28]*

3.2 IoT Communication Protocols

In terms of memory, storage, and processing power, today's general-purpose computing devices, e.g., smartphones, laptops, and personal computers, are becoming remarkably powerful. In comparison, IoT devices tend to have more constrained computing resources. Partly because IoT devices are meant to be "cheap" and deployed en-masse for a specific application, thus packing high-performance computing hardware may inflate the cost of the device.

Moreover, high-performance computing parts often use more energy than the lower-performance ones, while IoT devices may have to operate in environments with minimal energy supplies. Although users may install IoT devices in their homes, many businesses deploy them at remote locations, often with poor network connectivity and power options. Even if installed indoors, it may not always be possible to plug IoT devices into a virtually unlimited power source such as a wall power socket. Thus, many IoT devices rely on limited energy sources such as batteries.

Users may install IoT devices in a mobile environment, e.g., a vehicle or a container loaded on a ship, where the device's location constantly changes and may pass through areas with low bandwidth and intermittent network connectivity.

Therefore, for some applications, developers have to design the hardware under limiting factors such as (i) low power, (ii) limited physical space, (iii) low bandwidth, and (iv) intermittent network connectivity –all while minimizing the costs. Such constraints may lead IoT manufacturers to put lower-performance computing resources in their devices to reduce energy consumption.

Limiting factors not only affect the choice of hardware; they also affect the software that runs on an IoT device. The software should be able to execute using the available computing resources, e.g., low memory, bandwidth, and CPU performance, and gracefully handle frequent network connectivity or power interruptions.

To this end, industry and researchers have proposed several protocols to operate in challenging environments of IoT devices. Message Queuing Telemetry Transport

(MQTT), Advanced Message Queuing Protocol (AMQP), and Constrained Application Protocol (CoAP) are among these protocols. Still, some developers may use other protocols, such as Hyper Text Transfer Protocol (HTTP). More sophisticated IoT devices may use a combination of protocols to support their different functionalities. For example, they may rely on Network Time Protocol(NTP) [122] for time synchronization. In this section, we describe MQTT, AMQP, HTTP, and CoAP, popular choices among IoT developers [123].

3.2.1 Message Queuing Telemetry Transport (MQTT)

MQTT is an application-layer, light-weight, publish-subscribe(pub/sub) protocol. It is designed for transporting messages in machine to machine(M2M/IoT) communication [124]. According to a survey of IoT developers by Eclipse foundation [123], MQTT was the top choice among the survey participants. MQTT is designed to operate in environments with low processing power and bandwidth and follows the client/server model. In this model, IoT devices are clients. An MQTT IoT client, e.g., a sensor, sends messages to an MQTT server called the broker. MQTT also has a secure version that works over Transport Layer Security (TLS). IANA has assigned port numbers TCP/8883 and TCP/1883 to the secure and non-secure versions of MQTT, respectively. MQTT protocol specification has undergone several revisions, and the latest version is MQTTv5.

3.2.2 Advanced Message Queuing Protocol (AMQP)

AMQP is another open application layer pub/sub protocol for transporting messages between two processes over IP networks. While it is popular among IoT developers, it is not limited to IoT applications. Developers and organizations who wish to exchange messages between their business systems may also use this protocol. AMQP supports a non-secure and secure version which runs over TLS. The secure version, AMQPS, uses port 5671 (TCP/UDP), and the non-secure one uses 5672.

3.2.3 Hyper Text Transfer Protocol (HTTP)

Regarding traffic and usage, HTTP is one of the most widely used application layer protocols on the Internet [125]. IoT devices may use HTTP for several purposes. Some use it to send their telemetry data to the server, while others, e.g., Smart TVs, use it for video streaming. Some devices even host a small HTTP server, which provides a web interface for the users to configure the device. Some developers use HTTP to circumvent network firewalls configured only to allow outgoing connections of well-known protocols such as HTTP. The non-secure version of HTTP uses TCP/80, and its secure version, HTTPS, uses port 443.

3.2.4 Constrained Application Protocol (CoAP)

Standardized in RFC 7252 [126], CoAP is a transfer protocol designed for sending messages by devices with minimal computing resources. CoAP is based on HTTP but requires significantly fewer resources on the device and the network and offers multicast support. CoAP typically uses UDP 5683 for non-secure and 5684 for secure (TLS-based) communication.

3.3 Detecting IoT Devices

In IoT device detection, we detect *what* IoT device is *where* in the network. In this process, also known as IoT device fingerprinting, we identify an IoT device or its type based on its unique intrinsic or behavioral properties [127].

IoT device detection is an essential first step toward characterizing the IoT ecosystem. It allows network operators to keep track of devices connected to their networks and later aids them in identifying the malicious and misbehaving IoT devices [127].

The problem of IoT device detection has gained traction among researchers. They investigated it in diverse networks using different datasets. Network sizes can vary from small test beds to larger ones like campuses and service provider networks. They collect various datasets from IoT devices' radio signals in the physical layer to packet payloads in the application layer. They rely on one or a combination of active and passive measurement techniques to obtain these datasets. In active measurement, they send probes to the devices and capture their responses, while in passive measurements, they passively capture the network traffic or radio signals transmitted by the IoT devices. Since our datasets from the vantage points only contain network traffic data, we only consider the solutions that use network traffic of devices.

Active measurement solutions such as [17–26, 128], scan the IPv4 Internet and apply different techniques on the received responses to perform IoT device detection. Kumar *et al.* [27] deployed agents at home users' premises and scanned their private networks to infer the presence of IoT devices. Sivanathan et al. [24] relied on specific port numbers that may also be used for specialized industrial IoT systems [25], though the approach used cannot be easily extended to general-purpose IoT devices and smart home systems that utilize popular ports, e.g., 443, 80.

Active measurement techniques, though scalable, have several drawbacks: they (i) only report the behavior of targets in response to the probe packets, not the traffic as observed through passive monitoring [28], (ii) are intrusive to the end-users and raise privacy concerns if we deploy agents at end-users' premises, (iii) are challenging to identify devices that reside behind a Network Address Translation(NAT), (iv) cover almost exclusively IPv4 Internet and have scalability issues to cover IPv6 internet(due to the vast number of IPv6 addresses).

Passive measurement techniques like [32, 33, 35, 36, 129–132] use the passively-collected network traffic for IoT device detection. The authors in [32, 35, 129–131] use a broad range of network features from packet captures, or DNS traffic [33, 36] to train a machine learning model and detect IoT devices in a lab environment. They rely on testbed data [35, 132, 133], or tools for the active discovery of the household devices and their network traffic [134]. On a larger scale, Mazhar et al. [28] used instrumented home gateways to look at IoT traces from over 200 households in a US city. Their analysis revealed that while the IoT space is fragmented, few popular cloud and DNS services act as a central hub for most of the devices and their data.

However, to our knowledge, only a few works –and only concurrent or later than our studies– have applied their solutions *in the wild* and *at scale*, e.g., on the Internet or in large networks like ISPs. The main challenge is the *poor availability* and *low granularity* of data sources. The available data sources are often in the form of large volumes of passively collected, aggregated, and sparsely sampled data, e.g., NetFlow [30] and IPFIX [31]. Although authors in [39] identify IoT devices by observing passive DNS traffic, such methodologies often raise substantial privacy concerns as they require looking into the packet payloads. There have been efforts to understand IoT traffic patterns using data from transit networks [135], though it has been challenging to validate the derived signatures successfully. Guo et al. [38] proposed a method to identify IoT devices by observing unique IP addresses that the device contact. We contrast this methodology with ours in Section 5.6.

These related works indicate that for detecting and enumerating IoT devices at scale, we need scalable methodologies that (a) are not intrusive to the end-users, (b) do not rely on payload, and (c) handle sparsely sampled data.

3.4 IoT Backend Servers

IoT backend servers are essential parts of the IoT ecosystem. As IoT devices tend to have constrained resources and lack the resources to do all the processing locally, they may have to offload some of the processing to more powerful servers on the Internet. For example, an IP camera may not have enough storage to store large amounts of video files; hence it eventually has to store them on a server. Moreover, IoT devices rely on their backend servers to receive critical updates. Many IoT devices can be controlled remotely via the Internet, and implementing this functionality without using some backend server on the Internet is significantly challenging.

The growth of IoT applications and the demand for the backend infrastructure to support them have led to the emergence of **IoT Backend Providers**: companies that sell specialized IoT backend server infrastructure and related services to IoT developers. Large cloud providers have also entered this market and added the IoT backend as-a-service to their products.

Despite the importance of the backend servers in the security and operation of IoT applications, they are not extensively studied. Similar to the IoT device detection studies, we categorize the existing related work on IoT backend servers into Active Measurement and Passive Measurement approaches.

Active Measurement. Izhikevich et al. [20] perform active scanning campaigns and include IoT services that are often reachable on non-IoT ports. Several commercial and non-profit solutions periodically scan the Internet using a wide range of ports, including IoT standard ports and offer annotated datasets [21–23, 136]. These solutions typically use a successful IoT protocol handshake as an indicator of an IoT backend server. While they may find many IoT backend servers, they may not be able to detect the IoT backend servers which run multi-purpose protocols such as HTTP(S).

Active measurement campaigns are used by, e.g., used by Srinivasa et al. [17] to detect IoT clients in the wild and characterize IoT device misconfigurations. Note that they explicitly look for devices and not the IoT backend servers. The same is valid for work that tries to identify the IoT devices that participate in attacks, e.g., the Mirai attack [9]. We conclude that most work using active scans focuses on IoT protocols, and the rest focuses on IoT devices and their security properties.

Passive Measurement. Passive measurement techniques primarily focus on the detection of IoT devices rather than the characterization of their backend servers. The ones focusing on backend servers lack the *scale*: they instrument a testbed [133] or a dataset from a few hundred home users [28]. In [133], researchers instrumented a testbed to obtain a full capture of IoT device traffic and analyzed the destinations that devices contact. They found that many IoT devices contact servers that do not belong to their manufacturers, and in most cases, they contact destinations outside of the device's region. In [28] the authors used instrumented home gateways to look at IoT traces from over 200 households in a US city. Their analysis revealed that while the IoT space is fragmented, few popular cloud and DNS services act as a central hub for most of the devices and their data.

The recent work on the IoT backend providers is even more limited. Alrawi *et al.* [40] pointed to the importance of studying the IoT backend providers. He *et al.* [137] developed a methodology to identify the traffic destined for IoT backend providers. One particular work performed a security vulnerability analysis of 10 backend providers and found exploitable flaws in their services [11]. Thus, IoT backend providers have not received much attention despite their importance. This has motivated us to study the infrastructure of these companies in Chapter 7.

3.5 ISP and IXP Vantage Points

We intend to perform a data-driven characterization of the IoT ecosystem. We collaborated with an ISP and an IXP in our studies. This section provides the details of each vantage point and the ethical considerations for handling their datasets.

3.5.1 European Tier-1 Internet Service Provider

In Chapters 4 - 7, we collaborated with a large European ISP network. As a vantage point, it provides datasets for our studies. Besides being a Transit Tier-1 network, it serves 15 million broadband subscriber lines. It uses NetFlow-v9 to export flow captures and to sample traffic in the ingress direction of its routers. We obtained access to the NetFlow-v9 streams from all of the routers. In addition to its massive scale, the collaboration with an ISP allows observing the traffic from almost all subscriber lines in both directions(sent and received). In contrast, in vantage points such as IXPs, for each member, we may only observe the traffic for the portion of the traffic forwarded via IXP (due to possible asymmetry between the forward and backward traffic path of the traffic).

Ethical Considerations Studying traffic data from ISPs may raise ethical concerns as it may be considered an analysis of customer activities. However, it is not the goal of our studies. In each chapter, we provide further details about its ethical considerations. Nevertheless, we undertook the following steps shared across all of our studies.

The data is processed in situ and on the ISP's premise. Following best operational practices, the flow data is deleted at an expiration date set at the data collection time. For our analysis, no data is copied, transferred, or stored outside the dedicated servers that the ISP uses for NetFlow analysis.

Moreover, since parts of the Netflow data can be used as Personal Identifiable Information (PII) for subscriber lines, they are anonymized. More specifically, the data is anonymized by the BGP prefix before it hits the disc. We also note that to minimize spoofing, the ISP uses best common practices, including network ingress filtering according to BCP38 [138].

3.5.2 European Internet Exchange Point

We provided a detailed introduction of IXPs in Section 2.1.3. In our studies in Chapter 4 and 5, we also collaborated with a large European IXP. It is one of the largest IXPs in the world, with more than 900 members and peak traffic of 12 Tb/s as of July 2022. It utilizes IPFIX (sees Section 2.2) to sample the traffic in the ingress direction. We obtained access to IPFIX streams, which are backward compatible with NetFlow-v9 and do not contain packet payloads.

The IXP's dataset augments our studies by adding visibility over the traffic from other members, including multiple ISPs. However, while massive, the IXP dataset, unlike ISPs, may not contain all the traffic in all directions of all subscribers. Indeed, due to traffic asymmetry on the Internet, the forward and backward traffic between a source-destination on Internet may not cross the IXP. Even in a single direction, only some portion of traffic may pass through the IXP. Thus, the IXP's flow captures may have a partial view of traffic of a member.

Ethical Considerations

Like ISP datasets, studying traffic data from IXPs may raise ethical concerns as it may contain PII. Following best operational practices, the IPFIX data is anonymized and is deleted at an expiration date set at the data collection time. For our analysis, no data is copied, transferred, or stored outside the dedicated servers that the IXP uses for IPFIX analysis.

3.6 Ethical Considerations for Other Datasets

Besides the passive network flow captures from ISP and IXP, we performed active scans, resolved DNS queries, and obtained external datasets in our studies. This part explains our ethical consideration steps for handling such datasets.

3.6.1 Active Scanning

In our methodologies in Chapter 7, we performed active scanning. We took care to minimize any potential harm to the operation of routers and networks. First, the measurement load was very low, i.e., a single packet per destination. We also performed a randomized spread of load at each target IPv6 in the hit list. Moreover, we coordinated with our local network administrators to ensure that our scanning did not harm the local or upstream network.

For the active scanning, we use best current practices [136, 139, 140] to ensure that our prober IP address has a meaningful DNS PTR record. We run a Web server with experiment and opt-out information that responds to the resolution of the DNS PTR domain. We did not receive complaints or opt-out requests during our active experiments.

3.6.2 DNS Resolution

Our methodology in Chapter 7 required active DNS resolutions. We took the following ethical considerations: We made sure that the load in the DNS resolvers is low, i.e., we allow ten seconds before subsequent resolution, and we utilize all the available resolvers. To perform these resolutions, we used three locations, two in Europe and one in the The United States. All the locations were well connected to the Internet. Our resolutions added negligible additional load to the network.

3.6.3 External Data

In Chapter 5 and 7, we applied for research accounts to both Censys and DNSDB. The accounts allowed us to query and download the data that had been collected, i.e., active IP and port scans, TLS certificates, and passive DNS requests and responses.

3.7 Chapter Summary

This chapter is based on all our studies in chapters 5 - 7. We defined the terms and described concepts important for understanding IoT-related studies. An ISP and an IXP are our two main vantage points: we provided the details about their structure and datasets. More importantly, we explained the ethical considerations to handle their large-scale datasets. We also investigated the related work on IoT device detection and IoT backend servers. We underlined the need for further research into IoT backend servers, especially given their important role in the operation and security of the IoT ecosystem.

4 Flowyager: Exploring Network-Wide Flow Capture Data

In Background chapter(Chapter 2), we introduced the problem of *a priori unknown network-wide queries* against large volumes of flow capture data. Our studies in upcoming chapters required processing such datasets. In addition, with the proliferation of IoT devices among millions of ISP subscribers, ISP operators can take advantage of systems. Given that exploited IoT devices can be weaponized and participate in large-scale coordinated Distributed Denial of Service(DDoS) attacks [141, 142], network operators have to continuously keep track of the activity in their networks over both long and short time windows. Over short time windows, e.g., minutes, network operators would like to identify and rectify unusual events e.g., attacks or network disruptions. Over long time windows, e.g., days or hours, network operators are interested in provisioning network capacity or making informed peering decisions. We also note that a system which supports these capabilities shall (i) be scalable (ii) reuse *existing* flow captures (iii) support *interactive* and *ad-hoc* queries, and (iv) support queries *across network sites* and *over time.*

In this chapter, we introduce Flowyager, a system built on top of existing voluminous network captures, which enables interactive data exploration. Toward this goal, 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. Next, 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. Then, we show that with Flowyager, the query response time for network-wide queries can be reduced from hours or minutes to seconds. Finally, 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 have made Flowyager and its code available for non-commercial use under the following link [143].

4.1 System Requirements

In this section, we devise the requirements of a system that should be able to answer *a priori unknown network-wide queries* with a fast response, enabling *interactive* exploration of network data *across network sites* and *over time*, as follows:

(1) Scalability: The system should grow with the network size, the number of data sources, and the analysis requirements. With this, it should enable distributed deployment and does 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 [144] 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) 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 time to be able to store more recent data, and (c) the international footprint with the requirement to comply with local legislation that may prohibit raw data transfer.

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.

	Network Monitoring	Data Analytics	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 4.1: *Comparison of systems w.r.t. functionality offered. ✓: full support, ✗: no support.*

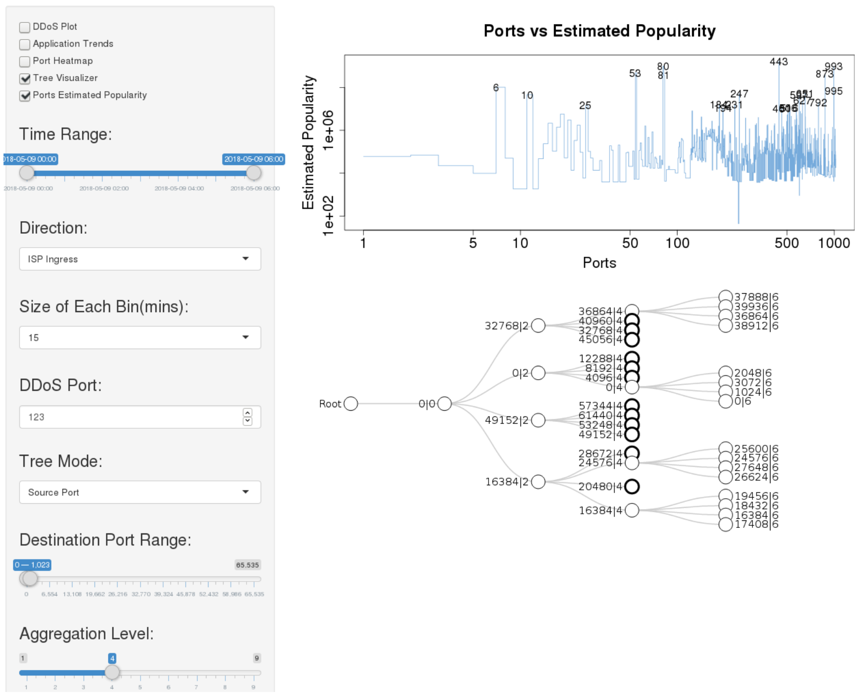
4.2 Flowyager Architecture

To address the requirements outlined in the beginning of 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 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. Recall the related approaches from 2.3.2, for a comparison between Flowyager and other approaches, we refer to Table 4.1.

To underline Flowyager's capabilities for exploring network data, we show in Fig 4.1 and Fig. 4.2 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.


☐ DDoS Plot
☐ Application Trends
☐ Port Heatmap
☑ Tree Visualizer
☑ Ports Estimated Popularity

Time Range: 2018-05-09 00:00 – 2018-05-09 06:00

Direction: ISP Ingress

Size of Each Bin(mins): 15

DDoS Port: 123

Tree Mode: Source Port

Destination Port Range: 0 — 1,023

Aggregation Level: 4

Figure 4.1: *Flowyager: Interacting with 1-feature Flowtrees.*

To better understand the system architecture, Figure 4.3 gives an overview of the overall system, while Figure 4.4 presents Flowyager's processing pipeline.

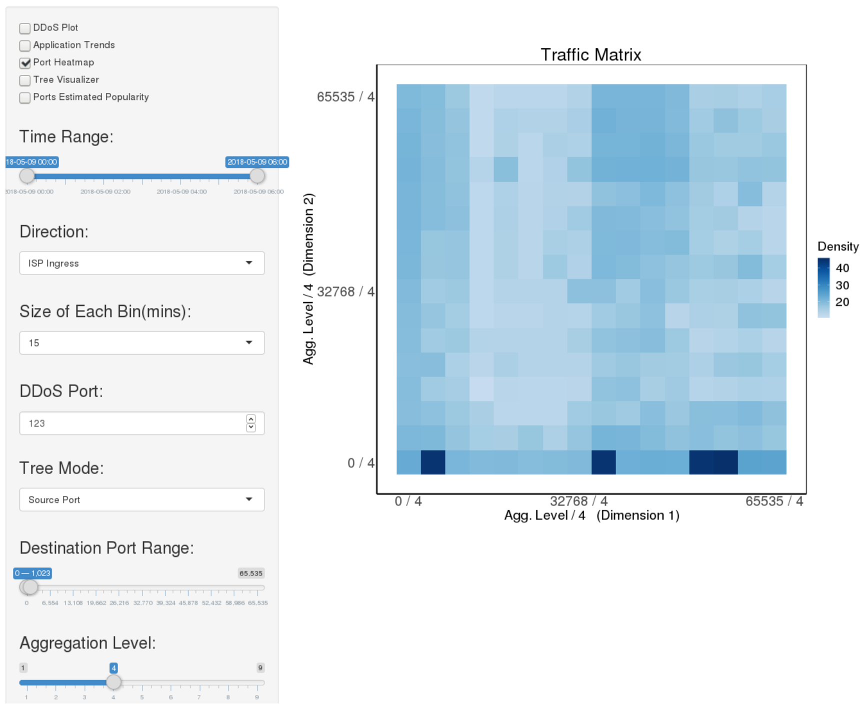
Each router sends its data to a NetFlow collector ①, which forwards it to one of potentially many distributed FlowAGG instances ②. Each FlowAGG instance computes summaries ③ and then uploads these either to another FlowAGG instance or directly to FlowDB ④[1]. FlowDB then processes the summaries ⑤ and 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 4.3.

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 [54] 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 4.4.

[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 [145, 146]



- [ ] DDoS Plot
- [ ] Application Trends
- [x] Port Heatmap
- [ ] Tree Visualizer
- [ ] Ports Estimated Popularity

Time Range: 2018-05-09 00:00 – 2018-05-09 06:00

Direction: ISP Ingress

Size of Each Bin(mins): 15

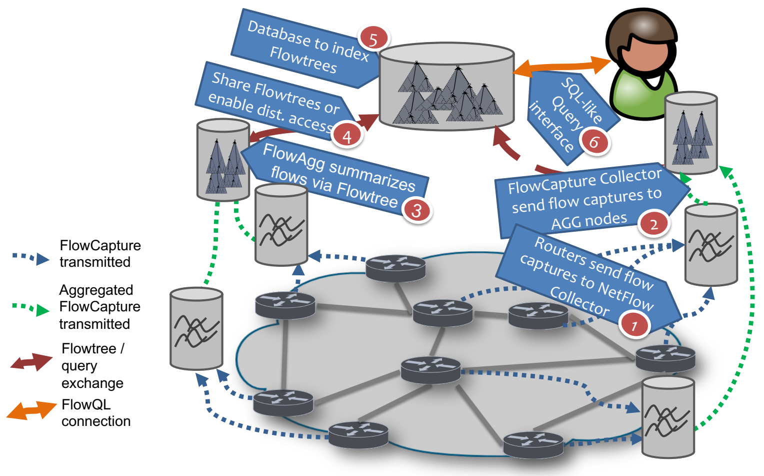
DDoS Port: 123

Tree Mode: Source Port

Destination Port Range: 0 — 1,023

Aggregation Level: 4

Figure 4.2: *Flowyager: Interacting with 2-feature Flowtrees.*



Database to index Flowtrees 5
Share Flowtrees or enable dist. access 4
FlowAgg summarizes flows via Flowtree 3
SQL-like Query interface 6
FlowCapture Collector send flow captures to AGG nodes 2
Routers send flow captures to NetFlow Collector 1
FlowCapture transmitted
Aggregated FlowCapture transmitted
Flowtree / query exchange
FlowQL connection

Figure 4.3: *Flowyager architecture.*

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.

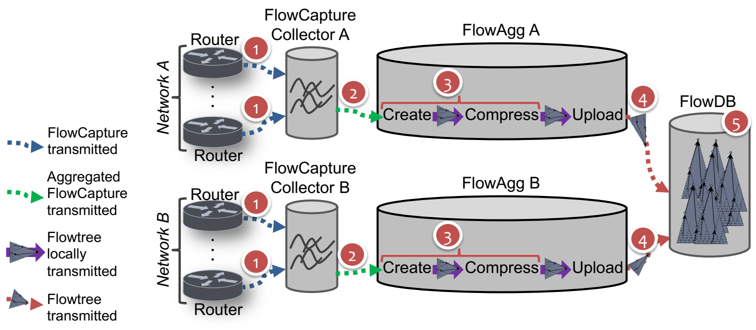


Figure 4.4: *Flowyager Processing Pipeline.*

4.3 Flowtree

Flowtree is the data structure 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.

4.3.1 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 five 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 [147–149] 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.

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 [111, 150]. 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

[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.

$O(H/\epsilon \log(\epsilon N))$, whereby N is the number of items processed, H is the number of hierarchy levels, and ϵ bounds the precision [111, 150].

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 4.5. 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 involving multiple features can benefit from 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 for identifying 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.

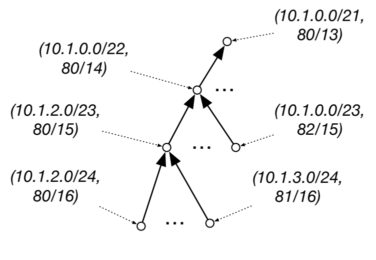


Figure 4.5: *Example: 2-Feature flow hierarchy.*

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))
```

4.3.2 Flowtree Data Structure

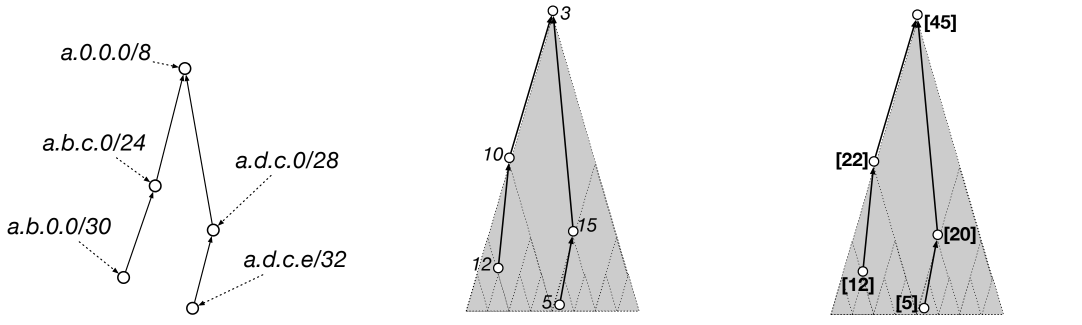
After evaluating different 1-d HHH data structures, including those of Cormode et al. [111, 150], Basat et al. [57], and Mitzenmacher et al. [112], we decided to augment the structure by Cormode et al.: this data structure is self-adjusting and its entries can be easily extracted via enumeration; thus, 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 4.7a and 4.7b. 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 4.6c.

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.

4.3.3 Flowtree: Visualizing the Concepts

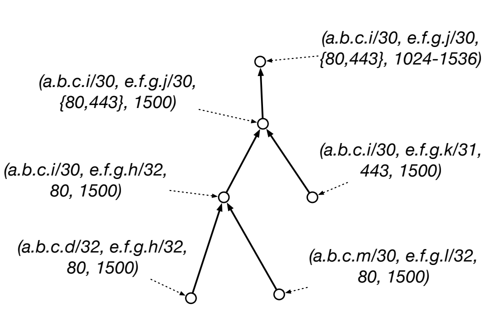
We start with the visualization of the differences between popularities and complementary popularities in Figure 4.7. Next, we show the two different feature hierarchies, namely a 1-feature hierarchy on IP addresses, and a 4-feature hierarchy on src/dst IP addresses and src/dst ports with and without popularities, see Figures 4.6a and 4.7.



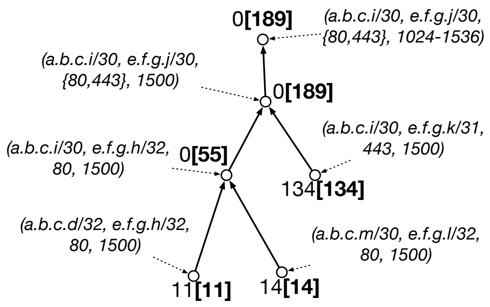
(a) *1-feature Flowtree: IP.* **(b)** *Comp_pop Flowtree.* **(c)** *Popularity in Flowtree*

Figure 4.6: *Flowtree concept.*

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 the 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 Algorithm 2. This uses two functions for finding the parents of a node. parent(node)

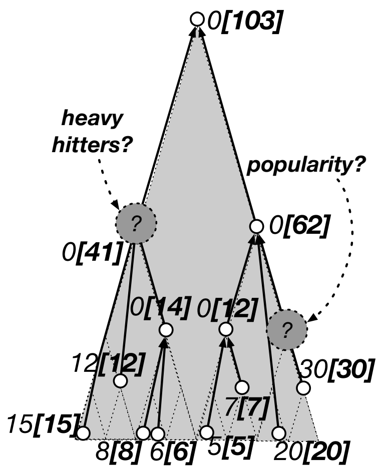

(a.b.c.i/30, e.f.g.j/30, {80,443}, 1024-1536)
(a.b.c.i/30, e.f.g.j/30, {80,443}, 1500)
(a.b.c.i/30, e.f.g.k/31, 443, 1500)
(a.b.c.i/30, e.f.g.h/32, 80, 1500)
(a.b.c.d/32, e.f.g.h/32, 80, 1500)
(a.b.c.m/30, e.f.g.l/32, 80, 1500)

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

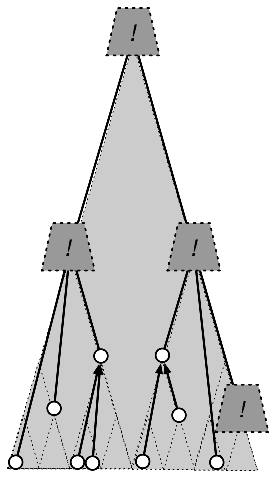

0[189]
(a.b.c.i/30, e.f.g.j/30, {80,443}, 1024-1536)
(a.b.c.i/30, e.f.g.j/30, {80,443}, 1500)
0[189]
(a.b.c.i/30, e.f.g.k/31, 443, 1500)
(a.b.c.i/30, e.f.g.h/32, 80, 1500)
0[55]
134[134]
(a.b.c.d/32, e.f.g.h/32, 80, 1500)
(a.b.c.m/30, e.f.g.l/32, 80, 1500)
11[11]
14[14]

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

Figure 4.7: *4-feature Flowtree.*



(a) *Queries:*
(a) Node is in Flowtree, is it a heavy hitter?
(b)Estimating the popularity of a node that is not in Flowtree



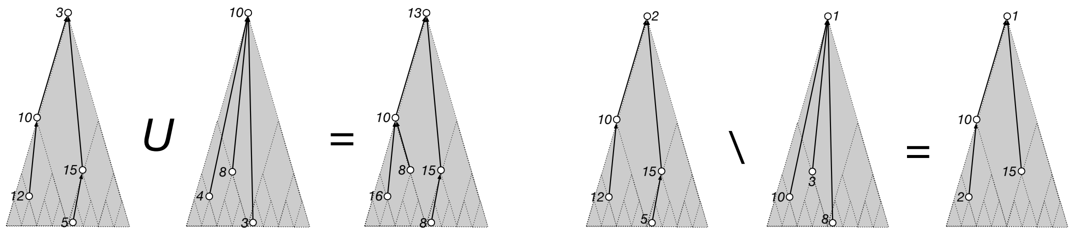
(b) *Above-t Query: k=29. The annotated nodes will be returned*

Figure 4.8: *Flowtree queries.*

refers 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 the Flowtree's memory footprint, we periodically, or on demand, delete nodes with low popularity. We first compute the popularities by using the stats function in Algorithm 2 and then prune nodes whose complementary resp. absolute popularity are below an adjustable threshold. This ensures that at any time, the number of



(a) *Merge* **(b)** *Diff*

Figure 4.9: *Flowtree Operators: Merge and Diff*

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 prevent 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.

4.3.4 Flowtree Operators

Query and drill-down: The base operators are *query* (see Figure 4.8) 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 $\text{comp\_pop}(p) + \sum_{c \in C_f} \text{pop}(c)$, see Figure 4.8. 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 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 Algorithm 3 and Figure 4.9a—, 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. 4.9b. Heavy changers are detected by using Top-k on the diff of the two trees.

Flowtrees maintain counters for various features of the flows. In the current implementation, we use counters for the 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.

4.4 FlowDB

FlowDB collects and stores Flowtree summaries computed by FlowAGG in persistent storage. Each Flowtree has a unique key 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 4.10 visualizes FlowDB's architecture.

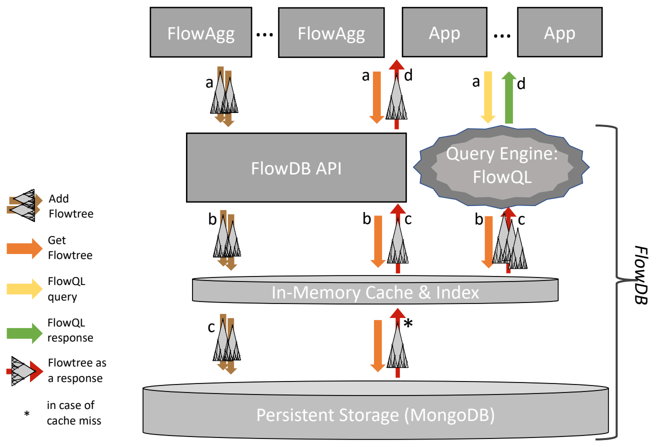


Figure 4.10: *FlowDB overview.*

4.4.1 FlowDB Implementation

Currently, our database of choice is MongoDB [151] because it is lightweight, although any other key-value datastore can be used. We use an in-memory index and an in-memory cache to accelerate query processing. 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 can receive multiple streams of Flowtrees from multiple FlowAGG daemons while answering queries to multiple users simultaneously. Parallelization is utilized 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 [152] 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.

4.4.2 FlowQL Query Language

To realize FlowQL, we took inspiration from SQL keywords, yet we developed our own grammar. We used ANTLR [153] to generate the parser for the grammar. We offer an interactive command-line shell and a graphical user interface using R shiny [154]–cf. the screenshots from Fig 4.1 and Fig. 4.2. 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

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

Table 4.2: *Deployment overview: IXP, ISP, and MAWI.*

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.

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

4.5 Experimental Deployments

We rolled out and tested Flowyager in three different types of networks, namely a large European IXP (IXP), a tier-1 ISP (ISP), and our testbed using a sample dataset (MAWI)—see Table 4.2 for an overview. In this study, 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 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

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 4.3: *Overview of the feature sets of Flowtree.*

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 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 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 [155] 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 and ISP datasets and 1 minute for the

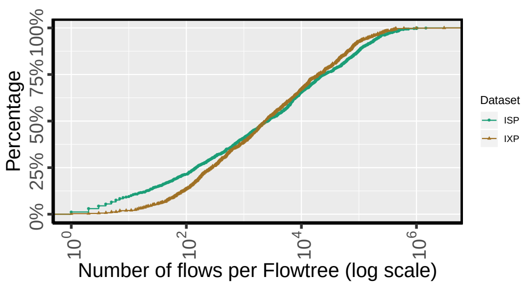


Figure 4.11: *ECDF of # of entries–all sites (IXP and ISP).*

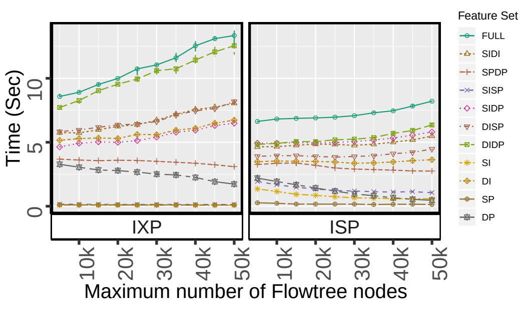


Figure 4.12: *Flowtree build time (IXP/ISP: four/one 15-min. trees) vs. max. # of nodes per feature set.*

MAWI 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 4.3 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.

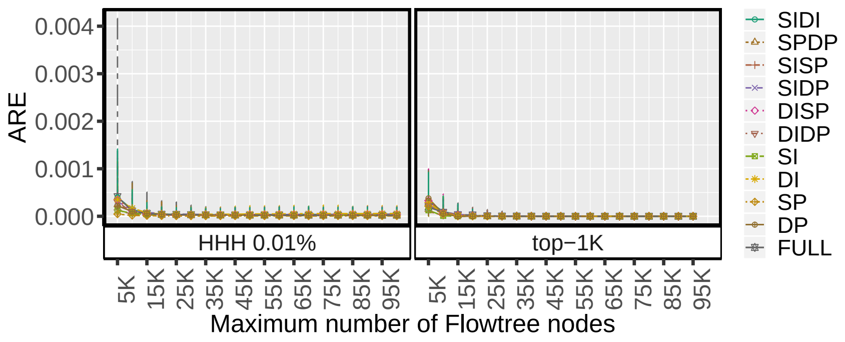
4.6 Flowyager Prototype Evaluation

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. Since these characteristics are related to our choice of underlying data structure and its resp. parameters, we start by evaluating Flowtree— the current basis of Flowyager.

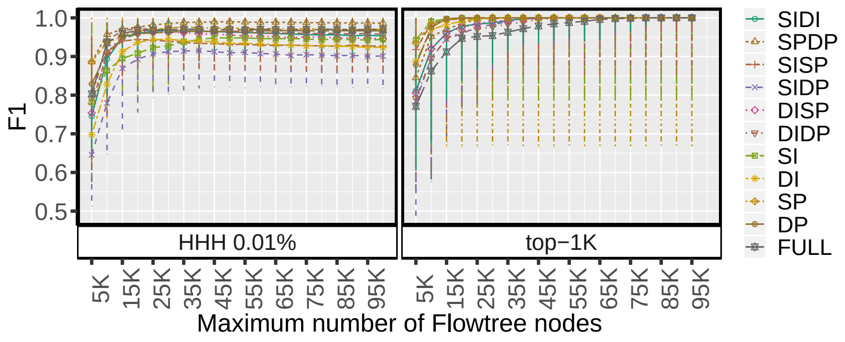
4.6.1 Flowtree Evaluation

Input data skewness: One motivation for using HHHs is to take advantage of the skewed input data. We indeed confirm that the flow captures are skewed in the sense that for all feature sets, all time periods, and all sites with enough traffic, the traffic volume follows a skewed distribution.

Next, the data structure should be able to summarize time periods with small as well as large numbers of flows as underlined by Figure 4.11, which shows the empirical cumulative distribution (ECDF) of the number of flow entries per 15-minute Flowtree for the IXP and the ISP datasets using a logarithmic x-axis. We find a huge skew. More than 37.5% of the time periods (per site) have less than 1,000 entries, yet more than 12.5% have more than 50k entries. This underlines that the data structure has



(a) *ARE vs. # of Flowtree nodes (all feature sets at IXP).*



(b) *F1-score vs. # of Flowtree nodes (all feature sets at IXP).*

Figure 4.13: *Accuracy of Flowtree for commonly-used queries (all feature sets at IXP).*

to be very flexible to efficiently summarize time periods with many as well as few flows.

Flowtree creation time: Next, we focus on the worst-case runtime to generate Flowtrees, which, in part, depends on the deployed hardware[3]. We focus on one hour of data, the busy hour, for the largest site at the IXP and 15 minutes of data–again busy hour and largest site, for the ISP to get an upper bound on the runtime. Note that the data includes more than 6.5M flows that have to be processed. We compute Flowtrees for each 11 feature set while varying the maximum number of Flowtree nodes from 5k to 50k. We repeat the experiments 10 times and measure the runtime, in terms of wall time, for generating trees as reported by the C++ chrono library[4].

Figure 4.12 shows the 10th and 90th percentile of the tree creation times vs. the maximum number of Flowtree nodes. All runtimes are well below 15 seconds for 1-hour resp. 15 minutes input files; thus, even if we have to process flows from 1,000+ sites, the deployed hardware, with moderate parallelization, is sufficient for generating all 11-feature Flowtrees in real time. In the worst case we needed 20 min to process traces from all 1,000+ sites over one hour; that is, Flowtree would only not become

[3]At the IXP we have Intel Xeon Gold 6148 CPUs; at the ISP we only have Xeon-E5-2650 CPUs
[4]We choose setup to similar to [57, 71] which also use wall time and preload the input data into memory.

a bottleneck if the throughput tripled and input from 1,000+ sites were to be processed. In that case, aggregating firs over different subsets of the flow space would be necessary. We notice different behavior for different features: The (destination IP, port) feature trees are very fast to compute, which can be explained by the fact that they exhibit the most skewed input distribution. The full (4-feature) trees take the longest—not surprising given that this feature combination potentially has the largest number of tree nodes.

We also notice that from one feature set to the next, the runtime sometimes decreases and sometimes increases as we increase the maximum number of tree nodes. The reasoning behind this surprising behavior is as follows. When the number of Flowtree nodes increases, while compressions happen less frequently, they take more time to run, given that they have to process a larger input. If the data is skewed, the increase of the compression runtime with the number of nodes is limited while the reduction in the average delay between two compressions is significant. Reversely, if the data is not less skewed, the increase in compression runtime outbalances the reduction in inter-compression delay.

Flowtree accuracy: Next, we look at the accuracy of the query results with focus on advanced queries, namely the 1-d HHH and top-K queries for Flowtrees with different featureset. Our metrics are the Average Relative Error, ARE, and the F_1 score. The ARE is the average of the ratios between the errors and the ground-truth values; that is, in our case, $\frac{1}{n}\sum_{i=1}^{n}\frac{|f_i-\hat{f}_i|}{f_i}$ with n the number of flows, f_i the flow popularity and $\hat{f}_i$ the estimated flow popularity. The F_1 score is the harmonic mean of precision and recall; accordingly, it accounts for both false positives and false negatives and ranges from 0 to 1—1 being the best value (perfect precision and recall) and 0 the worst. We calculate the ARE and the $F1$ score for the 1-d HHH and top-K queries, with thresholds of 0.01% and K=1000 respectively, for each 15-minute Flowtree and all sites over the IXP's busy hour, letting the maximum number of nodes in the Flowtrees vary from 5k to 100k. Note that we only evaluate the queries if a Flowtree summarizes at least 10k flows within the 15-minute time period since otherwise, the results would be a fraction of a flow, which does not exist. To generate the ground truth, we use a Flowtree with an unrestricted number of nodes. Finally, we only accept exact matches: in case of HHH, if a generalized flow f is in the actual heavy hitters it has to be returned by the HHH query; if the HHH query returns instead, a parent or child of f in the tree; this is a miss.

Figure 4.13a plots the median ARE values vs. the maximal number of nodes in the Flowtree and includes 10th and 90th percentiles as error bars in top-K and HHH queries. Our experiment shows that even for 10k Flowtrees the median ARE values are less than 0.0002 for all feature sets. Moreover, the main reason for ARE variations are flows with relatively small popularity.

The results for the $F1$ scores—see Figure 4.13b which shows the median together with the 10th and 90th percentile vs. the number of nodes per tree—confirm the excellent performance of Flowtree. Even for small trees, the median numbers are well above 0.9 for most feature sets. Moreover, the number of outliers is small.

Data structure	1k	5k	10k	20k	40k
Flowtree	.19 (.31)	.77 (.92)	.92 (.99)	.98 (.99)	.99 (.99)
RHHH	.42 (.11)	.50 (.57)	.91 (.92)	. 92 (.94)	.93 (.95)

Table 4.4: *F1 score on top 1k src (dst) IPs for 1k, 5k, 10k, 20k, and 40k node Flowtree and RHHH trees.*

Flowtree vs. RHHH: Next, we compare Flowtree to a state-of-the-art data structure, the constant time updates in hierarchical heavy hitters (RHHH) [57]. More precisely, RHHH is a randomized version of the deterministic HHH algorithm (dHHH) proposed by Mitzenmacher et al. [112]. RHHH has $O(1)$ update complexity, improving the $\Omega(H)$ update complexity of its deterministic counterpart, where H is the number of hierarchy levels.

While both Flowtree and RHHH take in the maximum node count as input, RHHH (and dHHH) have an additional input parameter: the HHH-threshold. The HHH-threshold determines if a frequent item is a heavy hitter, and, thus, if a node should be maintained in the tree. This complicates the usage of RHHH since neither the number of flows nor their popularity distribution is known in advance. Setting the threshold too high creates a very shallow tree with high aggregation, e.g., /16s and /8s, which does not keep enough detail. Setting the threshold too low may result in a tree with more nodes than the maximum node count. Indeed, we run into these limitations when executing the publicly available code [156] on the corresponding input. Hence, we evaluated the two systems under similar conditions, i.e., with the dataset that was used to evaluate RHHH [57] (CAIDA). The evaluation dataset comes from Equinix-Chicago trace of CAIDA [157]—this contains 20 Million packets (no sampling) from a 1Gbps link in the colocation facility in Chicago. In contrast, note that Flowtree is self-adjusting.

We used a number of metrics: (1) system runtime, (2) F1, on top 1k sources or destination of the input trace are present in the trees of 1k, 10k, 20k, and 40k nodes, (3) accuracy (ARE), i.e., how well the Flowtree or RHHH estimate the counters of the heavy hitters, either single IPs or aggregations.

The system runtime for creating RHHH trees is, as expected, quite constant: around 26 seconds. For Flowtree the time is higher, around 50 seconds, even as the number of nodes increases.

With regard to F1 score–identifying the correct set of heavy hitters–we find that if RHHH is not tuned, its performance is poor: very few of the IP heavy hitters are present and the trees are very small; the F1 of Flowtree is significantly better. Table 4.4 reports on the top-1k heavy hitter IPs indeed in the tree for Flowtree vs. RHHH with different total numbers of nodes(each time the threshold in RHHH is adjusted to produce 1k heavy hitters, i.e., the same output as Flowtree). For trees with up to 10k nodes, Flowtree includes a significantly larger number of heavy hitters than RHHH but beyond 10K nodes the differences get smaller.

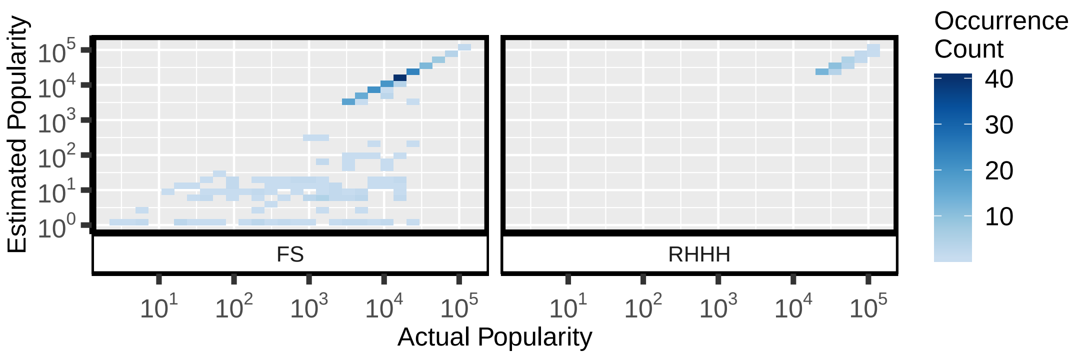


Figure 4.14: *Comparison of estimated vs actual popularities using Flowtree (left) and RHHH (right).*

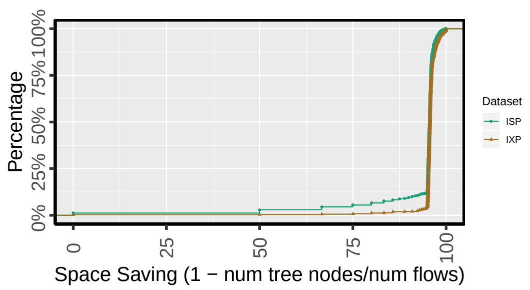


Figure 4.15: *ECDF of space-saving for all Flowtrees (all time intervals/IXP sites)*

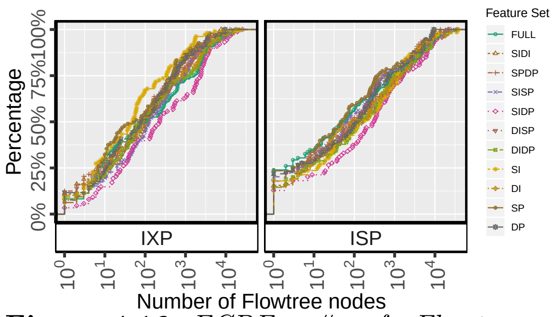


Figure 4.16: *ECDF: # of Flowtree nodes (IXP and ISP).*

Next, we turn our attention to the accuracy of the estimated values for each heavy hitter. We plot in Fig. 4.14 the estimated value using Flowtree (left) and RHHH (right) compared to the actual value for the 1k node trees—ARE on the top .1% IPs of 0.71 for Flowtree vs. 0.92 for RHHH.

The closer a point is to the diagonal the higher its accuracy. At first glance, RHHH might look better. However, it only contains a small subset of the relevant HHs as many top-1k entries are aggregated by RHHH. Thus, Flowtree again significantly outperforms RHHH.

Flowtree space saving: Given that we can compute Flowtrees efficiently and that they accurately answer 1-d HHH queries, we move on to study their space efficiency. Given the $F1$ scores and ARE values, we, for the rest of this chapter, choose 10k nodes for the 1-feature Flowtrees for src and dst ports and 40k nodes for all other feature combinations. (While 20k may be sufficient, using 40k does not increase the storage resp. communication overhead significantly, as we apply a final compress operation before using any Flowtree.)

To highlight the ability of Flowtree to compress its input, Figure 4.15 plots the ECDF of Flowtrees space saving ($1 - \frac{\texttt{\#nodes in tree}}{\texttt{\#input flows}}$) for all sites and all 15-minute time intervals. For almost all Flowtrees, the space savings are well above 95%. This is also underlined by Figure 4.16 which shows the ECDF of the number of actual

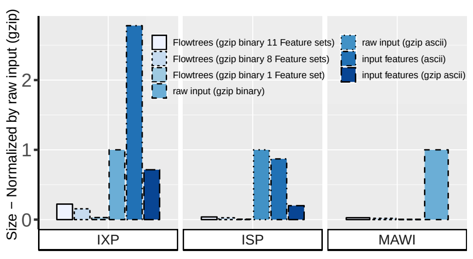
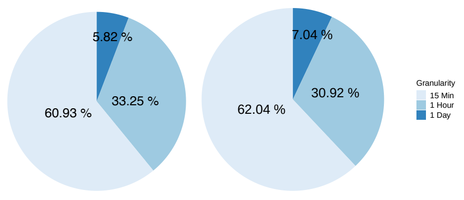


Figure 4.17: *Space usage vs. raw compressed (gzip) input data.*



(a) *IXP* **(b)** *ISP*

Figure 4.18: *Pie Chart: MongoDB footprint.*

Flowtrees nodes. Note that a Flowtree will always contain less than 40k/10k nodes because we always run a final compression. Alternatively, it might simply happen that the data did not contain enough different feature combinations in the first place.

4.6.2 Flowyager Evaluation

We evaluated Flowyager from space efficiency and query response time perspectives as follows:

Flowyager space efficiency: Given the above results regarding the capabilities of Flowtree, it is not surprising that Flowyager achieves excellent compression ratios. For the IXP (ISP), 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. For a visualization of the space efficiency relative to the size of the raw compressed (gzip) input data, see Figure 4.17.

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

	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

Table 4.5: *Benchmark queries for Flowyager evaluation. Note that these queries correspond to those identified in Table 2.1.*

Flowyager query response time 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

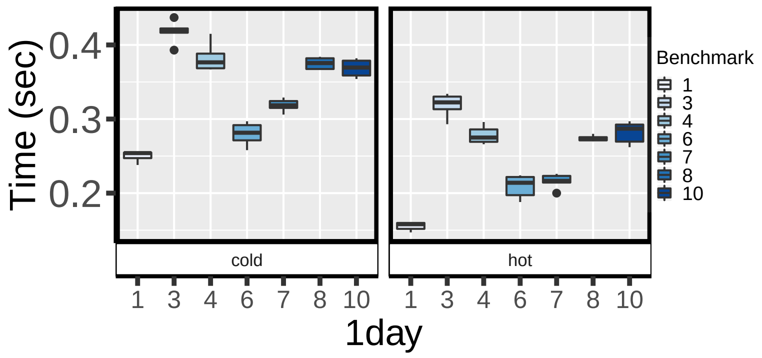


Figure 4.19: *IXP: Flowyager times, 1 Day aggregation Flowtrees. (See Table 4.5 for benchmarks).*

Table 2.1—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 4.5, 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 dataset. We evaluate answering using FlowQL with Flowtrees 1-day aggregation. On the IXP machine, we execute each benchmark ten times and measure, just as before, the wall time as reported by the C++ chrono library.

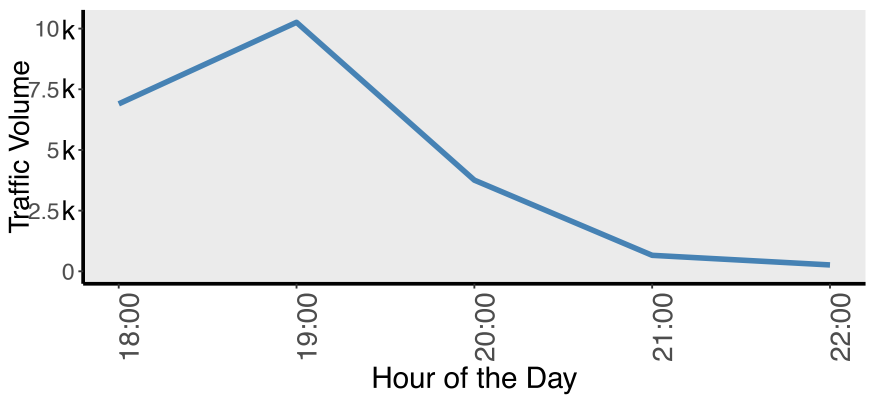
Figure 4.19 shows the resulting FlowQL query response times for each benchmark as boxplots. Hereby, we distinguish between cold and hot query response times. In the hot case, relevant Flowtrees may be retrieved from the in-memory cache. In the cold case, we restart the in-memory cache process for each benchmark. If we use the 1-day Flowtrees, see Figure 4.19, 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 found that the results are accurate[5].

4.6.3 Flowyager Limitations

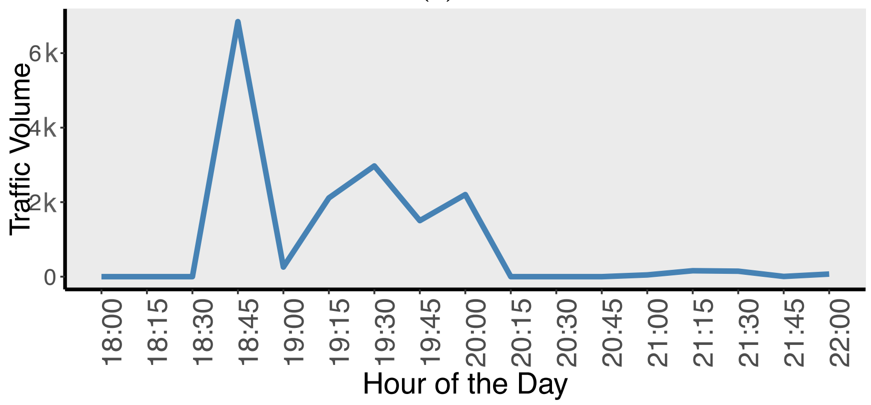
Flowyager is adaptive and supports HHH and physically distributed execution. We acknowledge that creating all Flowtrees does add some overhead–one day does take roughly 4 hours. However, this is a one-time operation, and overhead only matters if we consider archived data, but the Flowtrees can well be generated as the flow captures arrive, recall Section 4.6.1. 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.

[5]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.



(a)



(b)

Figure 4.20: *ISP: DDoS NTP attack investigation.*

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 systems such as ClickHouse [101] 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 [59].

4.7 Investigating DDoS attacks with Flowyager

Network attacks, and in particular, distributed denial-of-service (DDoS) attacks are an ongoing nuisance for network operators as well as network users. In this regard, compromised IoT devices are increasingly weaponized to launch large-scale coordinated DDoS attacks. A large body of research papers has focused on techniques for detecting DDoS attacks, see, e.g., [158–161], including references and citations. Indeed, the multitude and the impact of DDoS attacks, see, e.g., [162, 163], have given rise to a variety of different mitigation techniques, see e.g., [164, 165]. Still, detecting DDoS attacks reliably and diagnosing their root causes, is critical for starting countermeasures or taking future preventive 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 [162, 166–168]. 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 the 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 4.20 shows a DDoS amplification attack on 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 4.20a, within a second. Then, we drill down to the 15 minutes granularity to infer the start and end of the attack, see Figure 4.20b, 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 be used to trigger an alarm which may then blackhole the attack traffic, e.g., using a system such as Stellar [164] or traffic scrubbing systems [163]. 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 use 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 as the time the flow arrives 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 users' needs. 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.

4.8 Chapter Summary

In this chapter, we showed that network flow captures are widely available and are essential for network providers to monitor the security and health of their networks and steer their evolution. Especially given the growing number of deployed IoT devices and associated security risks, the risk of large-scale coordinated attacks by exploited IoT devices is higher than ever. Thus, ISPs and network operators need to be able to explore these captures to detect and investigate such attacks quickly.

Yet, due to the ever-increasing size and complexity of flow captures, their analysis is time-intensive and challenging. In the past, this has substantially hindered ad-hoc queries across multiple sites, for different time periods, and over many network features. In this chapter, we design, develop, and evaluate 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.

5 Detection of IoT Devices in the Wild

This chapter presents a methodology for detecting home IoT devices in-the-wild at an ISP, and an IXP, by relying on passive, sampled network traces and active probing experiments. We build on the insight that IoT devices typically rely on backend infrastructure hosted on the internet to offer their services. While contacting such infrastructure, they expose information, including their traffic destinations, even when a device is not in use [133]. One of the challenges of detecting IoT devices at such scale is the *poor availability and low granularity* of data sources. The available data is often in the form of centrally-collected aggregate and sampled data (e.g., NetFlow [30], IPFIX traces [31]). Thus, we need a methodology that (a) does not rely on payload and (b) handles sparsely sampled data.

Another challenge is *traffic patterns diversity*, across IoT devices and their services. (Here, we refer to IoT services as the set of protocols and destinations that are part of the operations of an IoT device.) We note that some devices, e.g., cameras, will generate significant continuous traffic; others, e.g., plugs, can be expected to be mainly passive unless used. Moreover, many devices offer the same service, e.g., the Alexa voice assistant [169] is available on several brands of smart speakers as well as on Amazon Fire TV devices. Here, the traffic patterns may depend on the service rather than the specific IoT device. Some services rely on dedicated backend infrastructures, while others may use shared ones, e.g., CDNs. Thus, we need a methodology that identifies which IoT services are detectable from the traffic and then identifies a unique traffic pattern for each IoT device and associated services.

Our key insight is that we can address these challenges by focusing our analysis only on the types of destinations contacted by IoT devices. Even with sparsely sampled data, the set of servers contacted by an IoT device over time can form a reasonably unique signature that is revealed in as little as a few hours. However, this approach has limitations; for example, we cannot use it to detect devices or services that use a shared infrastructure with unrelated services (e.g., CDNs).

To understand the detectability of IoT devices in the above-mentioned environment, we focus on the possible communication patterns of end-user IoT services and the types of destinations they contact. Figure 5.1 shows three possible communication patterns on top of a typical network topology. This includes three households, an ISP, as well as a dedicated infrastructure, and a CDN that hosts multiple servers. Device *A* is deployed by two subscribers and only contacts one server in the dedicated infrastructure. Device *B* is deployed by a single subscriber and contacts both a

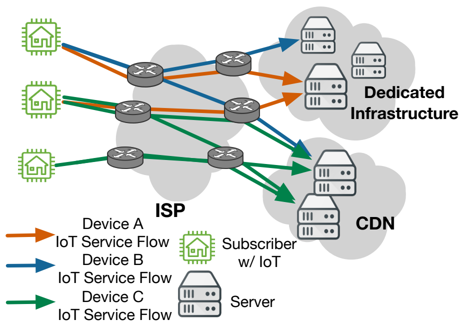

ISP
Dedicated Infrastructure
CDN
Device A IoT Service Flow
Device B IoT Service Flow
Device C IoT Service Flow
Subscriber w/ IoT
Server

Figure 5.1: *Simplified IoT communication patterns.*

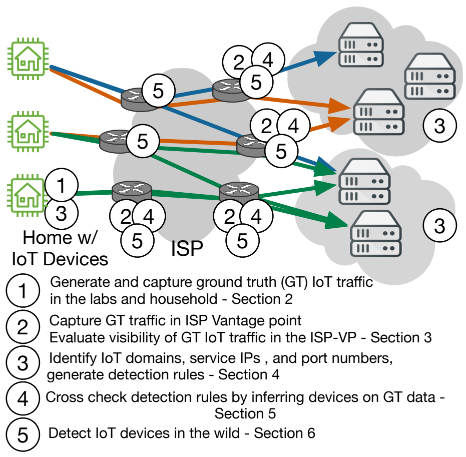

Home w/ IoT Devices
ISP
1 Generate and capture ground truth (GT) IoT traffic in the labs and household - Section 2
2 Capture GT traffic in ISP Vantage point Evaluate visibility of GT IoT traffic in the ISP-VP - Section 3
3 Identify IoT domains, service IPs , and port numbers, generate detection rules - Section 4
4 Cross check detection rules by inferring devices on GT data - Section 5
5 Detect IoT devices in the wild - Section 6

Figure 5.2: *General methodology overview.*

dedicated server as well as a CDN server. Device *C* is deployed by two subscribers and contacts only CDN servers. We observe that using NetFlow traces at the ISP edge, it is possible to identify subscriber lines hosting devices of type *A* and *B*. Devices of type *C* are harder to detect given the sampling rates and header-only nature of NetFlow.

In this chapter, we used a unique testbed and dataset to build a methodology for detecting and monitoring IoT devices at scale (see Figure 5.2). We first used controlled experiments, where we tunneled the traffic of two IoT testbeds with 96 IoT devices to an ISP. This provided us with ground truth IoT traffic within this ISP (Section 5.1). We confirmed the visibility of the ground truth IoT traffic using the NetFlow ISP data (Section 5.2). Next, we identified backend infrastructures for many IoT services from the observed ISP IoT traffic (Section 5.3). We augmented this base information with data from DNS queries, web certificates, and banners. Next, we used the traffic signatures to identify broadband subscriber lines using IoT services at the ISP, as well as an IXP (Section 5.5). Finally, we discuss our results, their significance, and limitations in Section 5.6, and conclude with a summary in Section 5.7.

5.1 IoT – Controlled Experiments

We need ground truth traffic from IoT devices, as observed both in a testbed and in the wild, for developing and testing our methodology. In this section, we describe our data collection strategy (see point (1) of Figure 5.2).

5.1.1 Network Setting

We utilize two *vantage points*, namely a large European ISP and a major European IXP.

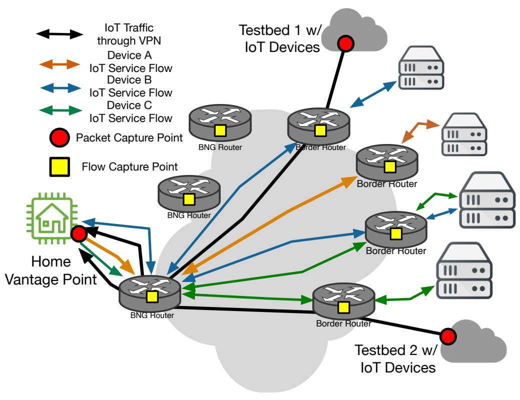


Figure 5.3: *ISP setup & flow collection points.*

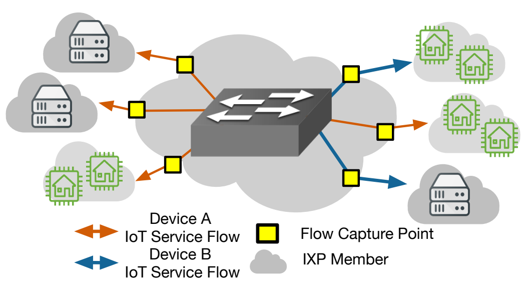


Figure 5.4: *IXP setup & flow collection points.*

ISP (ISP-VP). The ISP is a large residential ISP that offers Internet services to over 15 million broadband subscriber lines. The ISP uses NetFlow [30] to monitor the traffic flows at all border routers in its network, using a consistent sampling rate across all routers. Figure 5.3 shows where NetFlow data is collected.

IXP (IXP-VP). The IXP facilitates traffic exchange between its members. At this point, it has more than 800 members, including international, with peak traffic exceeding 8 Tbps. The IXP uses IPFIX [31] to collect traffic data across its switching fabric at a consistent sampling rate, which is an order of magnitude lower than the one used at the ISP. Figure 5.4 illustrates where the IPFIX data is collected.

Ethical considerations ISP/IXP. Neither the ISP nor the IXP flow data contain any payload data, thus no user information. We distinguish user IPs from server IPs and anonymize them by hashing all user IPs, following the method described in [170]. The address space of the ISP residential users is known. We call an IP a server IP if it receives or transmits traffic on well-known ports or if it belongs to ASes of cloud or CDN providers. The ports include, e.g., web ports (80, 443, 8080), NTP (123), and DNS (53). Moreover, we do not have any specific user activity and can only access and report aggregated statistics in accordance with the policies of the ISP and IXP.

Subscriber line (Home-VP) Network setup. In order to ingest ground truth traffic into the network, we need privileged access to a *home subscriber line.* For this, we use the ISP-VP, but rather than deploying all IoT devices directly within the home, we placed a VPN endpoint with an IP out of the /28 subscriber's prefix and used it to ingest IoT traffic tunneled to the server from two IoT testbeds, one in Europe, and one in the US, see Figure 5.3. The measurement points within the ISP will also capture this traffic. We simply excluded this traffic from our dataset, as the VPN tunnel endpoints are known to us, and for each experiment, we use the default DNS server for the ISP. Importantly, since the /28 prefix is used explicitly for our experiments, there was no other network activity other than that of the IoT devices.

Category	Device Name
Surveillance	Amcrest Cam, Blink Cam, Blink Hub, Icsee Doorbell, Lefun Cam, Luohe Cam, Microseven Cam, Reolink Cam, Ring Doorbell, Ubell Doorbell, Wansview Cam, Yi Cam, ZModo Doorbell
Smart Hubs	Insteon, Lightify, Philips Hue, Sengled, Smartthings, SwitchBot, Wink 2, Xiaomi
Home Automation	D-Link Mov Sensor, Flux Bulb, Honeywell T-stat, Magichome Strip, Meross Door Opener, Nest T-stat, Philips Bulb, Smartlife Bulb, Smartlife Remote, TP-Link Bulb, TP-Link Plug, WeMo Plug, Xiaomi Strip, Xiaomi Plug
Video	Apple TV, Fire TV, LG TV, Roku TV, Samsung TV
Audio	Allure with Alexa, Echo Dot, Echo Spot, Echo Plus, Google Home Mini, Google Home
Appliances	Anova Sousvide, Appkettle, GE Microwave, Netatmo Weather, Samsung Dryer (idle), Samsung Fridge (idle), Smarter Brewer, Smarter Coffee Machine, Smarter iKettle, Xiaomi Rice Cooker

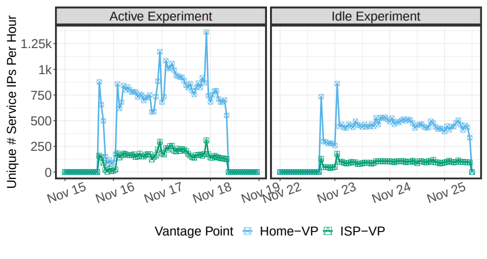
Table 5.1: ***IoT devices under test.*** idle *indicates that we capture the traffic just for idle periods because the experiments could not be automated.*

Ethical considerations–Home-VP setting. With the ISP's cooperation, we could use a reserved /28 allocated to this specific subscriber line (Home-VP) (with signed explicit consent) out of a /22 prefix reserved for residential users. Thus, the analysis in this study only considers traffic explicitly ingested by the ground truth experiments and does not involve any user-generated traffic.

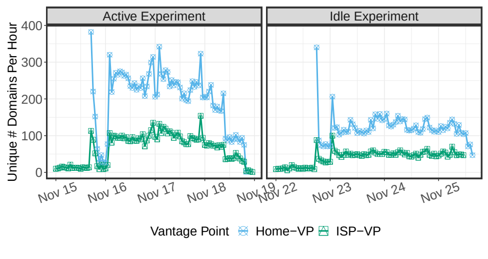
5.1.2 Ground Truth Traffic Setting

The IoT testbeds used here consist of 96 devices from 40 vendors. We selected the devices to provide diversity within and between different categories: surveillance, smart hubs, home automation, video, audio, and appliances. Most of these are among the most popular devices, according to Amazon, in their respective region. Our testbed includes multiple instances of the same device (56 different products) so that we can see the destinations that each product contacts in different locations. For a list of the IoT devices and each device category, we refer to Table 5.1. We redirect all IoT traffic to the Home-VP within the ISP, and we capture all the traffic generated by the IoT devices (see ① in Figure 5.2).

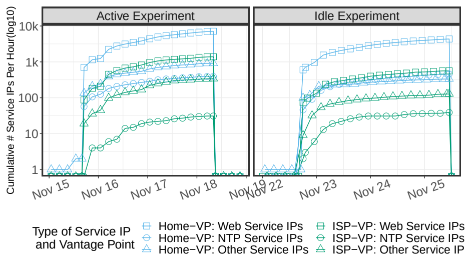
Most of the selected IoT devices are controlled using either a voice interface provided by a voice assistant (such as Amazon Alexa) or via a smartphone companion application. We use the voice interface to automate active experiments by producing voice



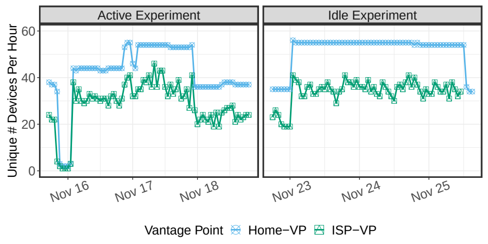
(a) *# Unique service IPs per hour.*



(b) *# Unique domains per hour.*



(c) *Cumulative service IPs per port.*



(d) *# Unique IoT devices per hour.*

Figure 5.5: *Home-VP vs. ISP-VP.*

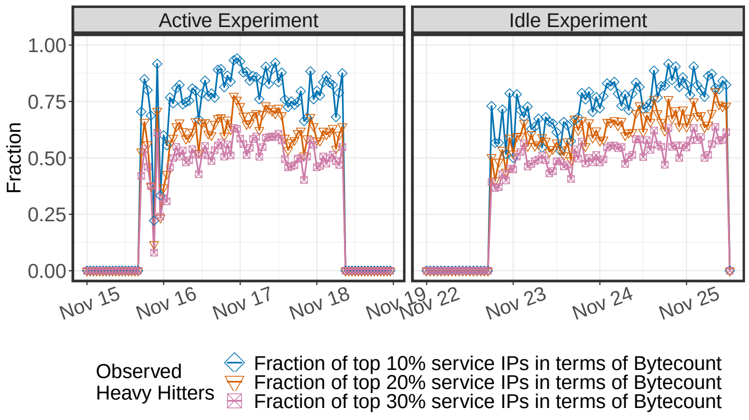


Figure 5.6: *Fraction of observed IPs ISP-VP vs. Home-VP per hour for popular servers (heavy hitters).*

commands using a Google Voice synthesizer. For IoT devices that support a companion app, we use Android smartphones, and we rely on the Monkey Application Exerciser for Android Studio [171] for automating simulated interactions between the user and the IoT device.

5.1.3 Active and Idle IoT Experiments

Our experiments can be classified into *idle* and *active* experiments.

Idle experiments. We define as *idle* the experiments during which the devices are just connected to the Internet without being actively used. We generate idle traffic for three days (November 23rd-25th, 2019) from both testbeds.

Active experiments. We define *active* experiments as those that involve automated interactions. We perform two types of automated interactions, each one repeated multiple times: (i) *power interactions*, since in a previous study [133] it was reported that many IoT devices generate significant traffic when they are powered off and on. We manage the power status of the devices through several TP-Link smart plugs that we can control programmatically, followed by two minutes of traffic capture; (ii) *functional interactions*, by automatically controlling the main functionality of the devices (i.e. the act of switching on/off the light for a smart bulb) via voice (either directly or through a smart speaker) or via a companion app running on a separate network with respect to the IoT device (to force the communication to happen over the Internet rather than locally). Unfortunately, some interactions for some devices cannot easily be automated (devices with *idle* in Table 5.1). For these devices, we consider only idle experiments. In total, we perform 9,810 active experiments between November 15th and 18th, 2019.

5.2 IoT Traffic – Visibility

In this section, we aim to understand (i) to which extent the IoT-related traffic of a *single subscriber line* reaches a diverse set of servers on the Internet, and (ii) whether the low sampling rate of NetFlow limits the subscriber/device visibility. For this, we rely on the ground truth traffic for the Home-VP. More specifically, we monitor the IoT traffic at both vantage points: the Home-VP, as well as the border routers of the ISP-VP (see (1) and (2) of Figure 5.2).

We first focus on the number of IP addresses that are contacted in each hour during the idle and the active experiments by the IoT devices, as stated in Section 5.1.3. We explicitly excluded DNS traffic since it is not IoT-specific. From Figure 5.5a, we see that during the active experiments, the IoT devices contact between 500 and 1,300 service IPs per hour when monitored at the Home-VP. Due to sampling, not all of this traffic is visible at the ISP-VP. We define *service IPs* as the sets of IPs associated with the backend infrastructures that support the IoT services. Indeed, the number of observed service IPs per hour in the ISP-VP decreases to an average of 16%. Overall, during our idle experiments, the total number of contacted service IPs was lower, but the average percentage of observed service IPs remained at 16.5%.

The spikes in the active experiments are partially due to power and functional interactions. This can be seen in the idle experiments, where the spike indicates the action of starting the device (only at the beginning). Note that these spikes are also visible in the sampled ISP NetFlow data.

At first glance, 16% sounds like a very small percentage. However, we note that the visibility of popular service IPs is significantly high. Figure 5.6 shows the fraction of service IPs that are visible for the servers contacted the most, according to byte count. For the top 10% of the service IPs, more than 75% are visible, rising up to 90% during some experiments. For less popular service IPs, e.g., the top 20% and top 30%, the visibility is only reduced to 70% and 60% in the active experiment and a bit lower for the idle experiment.

If we consider the entire period of our experiments, the percentage of visible service IPs is more than 34% and 28% for idle and active experiments. Overall, more than 95% of service IPs are visible at the daily level for the top 20%. Although we cannot observe *all* IoT devices activity at the ISP-VP, a significant subset is visible.

While any specific service IP may not matter that much for an IoT service, its communications with a server domain name that may be hosted on multiple service IPs is essential. From the Home-VP, we know which service IPs correspond to which domain. Thus, we can determine which observed service IPs at the ISP-VP belong to which domain. This information is relevant for our methodology because, in the ISP NetFlow data, only IPs are visible. Figure 5.5b shows the number of observed Fully Qualified Domain Names (FQDNs, we will refer to them as domains or domain names for the rest of the study) at the Home-VP and the ISP-VP. Many domains are hosted at multiple service IPs; hence we see that the number of observed service IPs is higher than the number of observed domains.

Figure 5.5d shows the number of observed IoT devices per hour from the ground truth IoT traffic. We observe a device when at least one packet from that device is seen within an hour. Note, For active mode, the experiments on devices from Testbed 1 (see figure 5.3), are initiated after Testbed 2. Therefore, all devices are not active during the same period. The average percentages of devices visible at ISP-VP during active and idle experiments are 67% and 64%, respectively.

Next, we separate the observable network activity by ports. More specifically, we consider Web Services (ports 443, 80, 8080), NTP services (port 123), and other services (the rest of the ports), and we show the cumulative number of service IPs contacted. The resulting plot, Figure 5.5c, shows that (i) the trend of observable service IPs at the Home-VP is mirrored at the ISP-VP, even when different services are considered, and (ii) the number of service IPs converges over time.

We also checked if any of the traffic from the Home-VP is visible at the IXP. However, neither during the active nor during the idle experiments did we observe traffic at the IXP. This is expected as the ISP is not a member of the IXP. Rather it peers directly (via private interconnects) with a large number of content and cloud providers as well as other networks.

In summary, our analysis of the ground truth IoT traffic shows that, despite the low sampling of NetFlow, popular domains, service IPs, and ports of a **single** subscriber line (the Home-VP) are visible at the ISP.

5.3 IoT Device detection methodology

In this section, we outline our methodology for detecting IoT devices in-the-wild. IoT services typically rely on a backend support infrastructure (see Figure 5.1) for user interactions. From our ground truth experiments, we noticed that this backend infrastructure is often also used for keep-alives, heartbeats, updates, maintenance, storage, and synchronization. This observation is consistent with previous works [28, 133].

We focus on identifying which Internet backend infrastructure is supporting *each* of the IoT devices that we deployed in our testbeds (see ③ in Figure 5.2). When we refer to Internet backend infrastructure, we use two different abstractions: (i) sets of IP address/port combinations as observable from the Internet vantage points, and (ii) sets of DNS domains. We also focus on domains because they are the primary indirect way for the devices to access their backend infrastructure. While domain names are typically part of the permanent programming of the devices, IP addresses are discovered during DNS resolution and may change over time.

A naive approach for identifying the backend infrastructure would be to use the ground truth traffic to identify which domains and, as a consequence, which service IPs are being contacted by each device. However, this is not sufficient for the following reasons:

Limited relevance of some domains: Not all domains are essential to support the services or are useful for classification; for example, some domains may be used for advertisements or generic services, e.g., `time.microsoft.com` or `wikipedia.org`, see Section 5.3.1.
Limited visibility of IP addresses: Since the ground truth data is captured at a single subscriber line only and DNS to IP mapping is rather dynamic, just looking at this traffic is not sufficient, see Section 5.3.2.1.
Usage of shared infrastructure: Not all IoT services are supported by a dedicated backend infrastructure. Some rely on shared ones, such as CDNs. In the former case, they can still have dedicated IP addresses; in the latter cases, they use shared IP addresses, see Section 5.3.2.1.
Churn: DNS domain to IP address mappings are dynamic, see Section 5.3.2.1.
Common programming APIs: Multiple IoT services may use the same common programming API or may be used by different manufacturers; as a result, they often rely on the same infrastructure. This is the case for relatively generic IoT services such as Alexa voice service. While this IoT service is available on dedicated devices, e.g., Amazon Echo, it can also be integrated into third-party hardware, e.g., fridges and alarm clocks [169]. We cannot easily distinguish these from network traffic observations.

Below we tackle these challenges one by one. The outcome is an IoT dictionary that contains mappings for individual IoT services to sets of domains, IP addresses, and ports. Based on IoT services, we generate rules for IoT device detection. For an overview of the resulting methodology, see Figure 5.7.

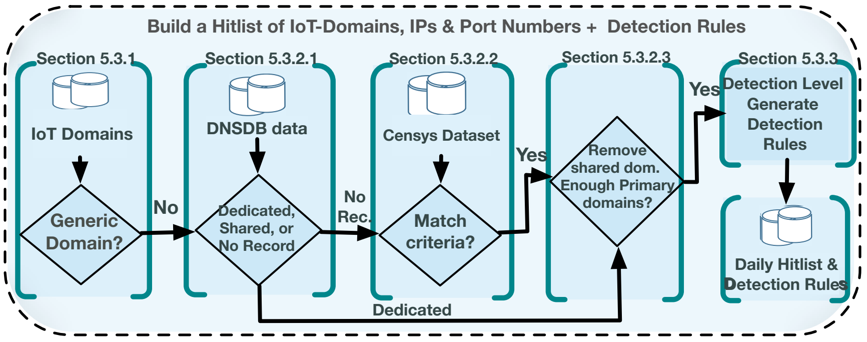

Build a Hitlist of IoT-Domains, IPs & Port Numbers + Detection Rules
Section 5.3.1
IoT Domains
Generic Domain?
No
Section 5.3.2.1
DNSDB data
Dedicated, Shared, or No Record
No Rec.
Section 5.3.2.2
Censys Dataset
Match criteria?
Yes
Section 5.3.2.3
Remove shared dom. Enough Primary domains?
Yes
Section 5.3.3
Detection Level Generate Detection Rules
Daily Hitlist & Detection Rules
Dedicated

Figure 5.7: *IoT Traffic detection methodology overview.*

5.3.1 Classifying IoT Domains

The amount and frequency of network traffic that an IoT device exchanges with its backend infrastructure varies from device to device; it depends on the complexity of its services, implementation specifics, and device usage. We highlighted this in Figure 5.8, where we show the average number of packets per device and per domain (using a log y-scale) for 13 different devices (a subset of devices) in their idle mode. The first observation is that most devices are supported by their own set of domains, and for many IoT services, this is a small set containing less than 10 domains. We refer to these as *small domain sets* as they correspond to *laconic* devices. Other devices *gossip* and have *sizable domain sets*. Figure 5.8 shows the domains of two example gossip devices (Apple TV in gray and Echo Dot in orange) and several laconic devices (rest of the colors).

Having a sizable domain set often indicates the usage of a larger infrastructure, which may not be dedicated to a specific IoT service. We find that most of these domains are mapped via CNAMEs to other domains. For the two gossiping examples considered in Figure 5.8, the domains of Echo Dot are mostly mapped to its own infrastructure. However, the ones of Apple TV are mainly mapped to a CDN– in this case, Akamai— that offers various services.

Based on these observations from our ground truth data, we classify the domains as follows:

IoT-Specific domains. Grouped into (i) *Primary* domains: registered to an IoT device manufacturer or an IoT service operator; and (ii) *Support* domains: that are not necessarily registered to IoT device manufacturers or service operators, but offer complementary services for IoT devices, i.e. *samsung-*.whisk.com* for Samsung Fridges, here whisk.com is a service that provides food recipes and images of food.
Generic domains. Domains registered to generic service providers that are heavily used by non-IoT devices as well, e.g., *netflix.com*, *wikipedia.org*, and public NTP servers.

We classify each domain name from our idle and active experiments using pattern matching, manual inspection, and by visiting their websites and those of the device manufacturers. Since the *Generic* domains cover non-IoT traffic, we do not further

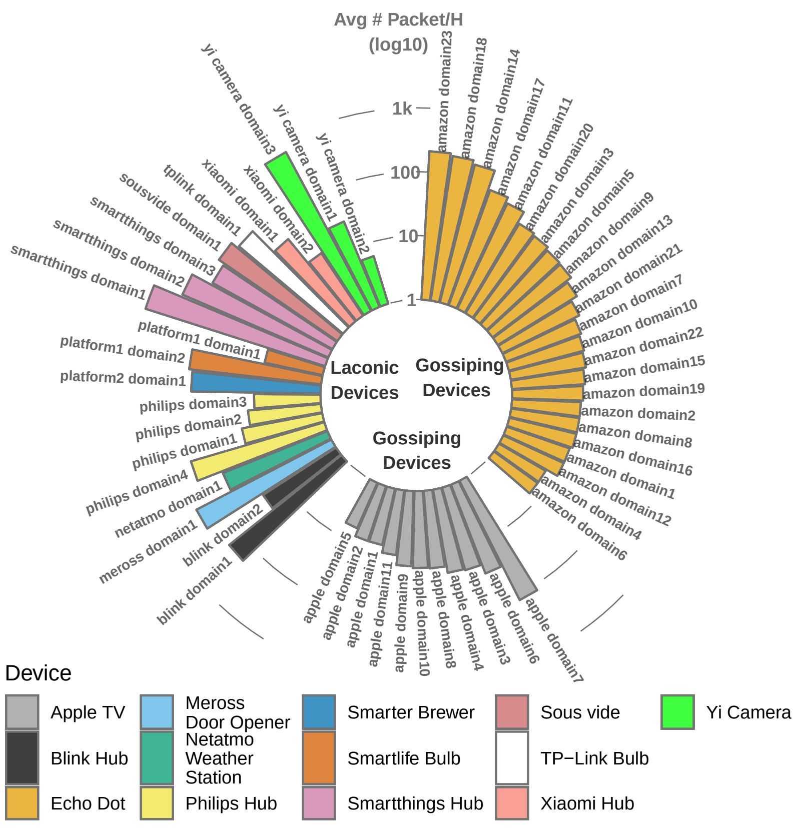


Figure 5.8: *Home-VP: Circular bar plot of average # of packets/hour per domain (log y-scale). The domains belong to 13 IoT devices and are separated into three groups: one for laconic and two for gossiping devices (Echo Dot and Apple TV).*

consider them. Rather, we focus on the *IoT-Specific* domains. As a result, we classify 415 out of the 524 domains as *Primary* and 19 as *Support* domains.

Next, we explore the volume of traffic the IoT devices exchange with all domains. Figure 5.9 shows the ECDF of the average number of packets per hour per domain for all IoT-Specific domains for both the idle and the active experiments. First, we note that almost all devices and domains , except for one device in its idle mode, are exchanging at least 100 packets per hour, and this *may not suffice for detecting* them in any given hour in the wild due to sampling. However, during the active experiments, we see that some domains are only used when the device is *active* or other domains receive significantly more traffic, up to and exceeding 10K packets, which *may suffice for detection.* These latter domains may be ideal candidates for detecting such devices in the wild.

5.3.2 Identifying Dedicated Infrastructures

Once we have a list of IoT-Specific domains (FQDNs) with their associated service IP addresses and port mappings from the ground truth experiments, we need to

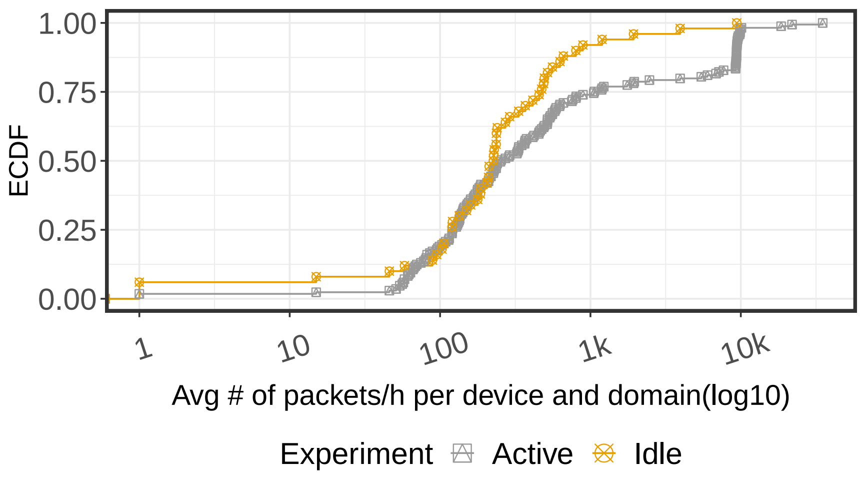


Figure 5.9: *Home-VP: ECDF of average # of packets/hour for all IoT-Specific domains, per device (idle and active experiments).*

understand whether they have a shared or dedicated backend infrastructure. The reason is that if we want to identify IoT services and consequently IoT devices in the wild by using network traces such as NetFlow, we can only observe standard network-level features such as src/dst IP and port numbers without packet payload. Therefore, if a service IP belongs to a shared infrastructure such as a CDN or a generic web hosting service, it can serve many domains, and it is impossible for us to exactly know which domain was actually contacted. To this end, the purpose of this section is two-fold. First, to expand the candidate service IPs beyond those directly observed in the ground truth experiments (to mitigate that, we are focusing on a single subscriber line). Second, to classify domains into those that use backend services hosted on dedicated infrastructure service IPs vs. those that rely on shared infrastructure service IPs. We do this by relying on DNSDB [172], Censys [173], and applying additional filters.

5.3.2.1 From IoT-Specific Domains to Service IPs: DNSDB

We use IoT-Specific domains to identify the backend infrastructure that is hosting them. To this end, we leverage the technique in [174] and use these domain names to identify all associated service IPs on which these domains are hosted during the time period of our experiments. We use both the ground truth experiments and external DNS databases, including DNSDB [175]. We found that the specific IP addresses to specific domain mappings can change often. However, DNSDB provides information for all domains served by an IP address in a given time period and vice versa; hence it mitigates the issues caused by this churn. DNSDB also provides all records for a given domain, including CNAMEs that may have been returned in the DNS response. Thus, we use DNSDB to check if a service IP address is *exclusively* used for a specific IoT service or if it hosts additional domains. We say a service IP is *exclusively* used if it only serves domains from a single "second-level" domain (SLD) and its

CNAMEs. However, we note that the CNAMEs may not involve the same second-level domain. Let us consider an example: the domain `devA.com` is mapped via a chain of CNAMEs such as `devA-VM.ec2compute.amazonaws.com` to IP `a.b.c.d`. This IP only reverse maps to `devA-VM.ec2compute.amazonaws.com` and its associated CNAME `devA.com`. Since this is the only CNAME associated with the IP, we may consider this IP a direct mapping for the domain. Yet, at the same time, we find support that public IP addresses assigned to a cloud resource such as a virtual machine in AWS EC2, occupied by a tenant, are not shared with other tenants unless the current resource is released. This is a popular service offered by multiple platforms [176–178]. Let us consider a second example: domain `devB.com`. It may use the Akamai CDN. Thus, the domain `devB.com` is a CNAME for `devB.com.akadns.net`. This domain then maps to IP `a.b.c.d`. However, in this case, many other domains, e.g., `anothersite.com.akadns.net`, also map to this IP. Thus, we may conclude that this domain is hosted on a *shared* infrastructure.

Once we understand if an IP is exclusively used for a specific IoT service, we can also classify the domains as either using a *dedicated* or *shared* infrastructure. For the former, all service IPs have to be dedicated to this domain for all days; otherwise, we presume that the domain relies on a shared infrastructure.

Once we apply this methodology to all 434 domain names, we find that 217 are hosted on dedicated service IPs, while 202 rely on shared backend infrastructure. For 15 of the domains, we did not have sufficient information in DNSDB. We handle them in the next step.

5.3.2.2 From IoT-Specific Domains to Service IPs: Censys

Among the reasons that DNSDB may not suffice for mapping some domains to service IPs is that (a) frequent remapping of domains to IPs or (b) missing data since the requests for the domains may not have been recorded by DNSDB, which intercepts requests for a subset of the DNS hierarchy. To overcome this limitation, we rely on the certificate and banner datasets from Censys [173], to infer the ownership of the domains and the corresponding IPs, as long as these are using HTTPS. For example, we did not find any record for the domain `c.devE.com` in the DNSDB dataset. We then check if device E uses `HTTPS` to communicate with this domain. This allows us to query for all service IPs that potentially offer the same web certificate as the hosts in this domain. For a certificate to be associated with a domain, we require that the domain name and the *Name* field entry in the certificate match at least the SLD or higher, i.e., the Name field of the certificates matches the pattern `c.devE.com` or `*.devE.com` and that there is no other *Subject Alternative Name (SAN)* in the certificate. Next, we query the Censys dataset for all IPs with the same certificate and `HTTPS` banner checksum for the domain from our ground truth dataset within the same period. This allows us to identify data for 8 out of 15 domains belonging to 5 devices.

5.3.2.3 Removal of Shared IoT Backend Infrastructures

In the last step of our methodology, we filter out devices that use shared backend infrastructures. We find that Google Home, Google Home Mini, Apple TV, and Lefun camera all have a shared backend infrastructure. For LG TV, we are left with only one out of 4 domains; for Wemo Plug and Wink-hub, we could not identify sufficient information. Because of this, we have excluded these devices from further consideration.

The result forms our *daily* list of dedicated IoT services, along with their associated domains, service IPs, and port combinations.

5.3.3 IoT Services to Device Detection Rules

Once we have identified the set of IoT services that can be monitored, we generate the rules for detecting IoT devices. Depending on the set of IoT services contacted by the devices, we can generate device detection rules at three granularity levels: (i) Platform-level, (ii) Manufacturer-level, and (iii) Product-level, from the most coarse-grained to the most fine-grained, respectively. In this section, first, we show how we determine the detection level for each device. Then, we explain how we generate the detection rules for each IoT device for the detection level that can be supported.

5.3.3.1 Determining IoT Detection Level

Platform-level: Some manufacturers use off-the-shelf firmware or outsource their backend infrastructure to IoT platform solution companies such as Tuya [179], electricimp [180], AWS IoT Platform [181]. These IoT platforms can have several customers/manufacturers that rely on their infrastructure. Therefore, we may be unable to distinguish between different manufacturers from their network traffic.
Manufacturer-level: The majority of our studied IoT services rely on dedicated backend infrastructures that the manufacturers themselves operate. We also observe that many manufacturers rely on similar APIs and backend infrastructures to support their different products and services. This makes distinguishing individual IoT products from their network traffic more challenging.
Product-level: This is the most fine-grained detection level, where we can distinguish between different products of a manufacturer, e.g., Samsung TV, or Amazon Echo vs. Amazon Fire TV. For detection at the product level, we underline the importance of side information about the purpose associated with a domain. With this information, we can improve our classification accuracy. For example, for Alexa Enabled devices, the domain `avs-alexa.*.amazon.com` is critical, as it is the base URL for the Alexa Voice Service API [169] (shown in Figure 5.8 as amazon domain23). Other examples are the Samsung devices that use the domain `samsungotn.net` to check for firmware updates [182].

Additionally, some advanced services of the devices often require additional back-end support from manufacturers. These may then contact *additional* domains. By considering more specific features (domains), the capabilities to distinguish products increases. We leverage these specialized features e.g., to distinguish Amazon Fire TV, which contacts significantly more domains than other Amazon products, e.g., Echo Dot.

5.3.3.2 Generation of Detection Rules

For any of our three levels of detection, we require that a subscriber contacts at least one IP/port combination associated with a Primary domain of the IoT service to claim detectability of IoT activity at the subscriber. However, if there are many domains, requiring only one such activity may not have enough evidence. For example, by monitoring a single domain, we can detect all Alexa Enabled devices, but this service can also be integrated into third-party hardware. Therefore, to detect products manufactured by Amazon, e.g., Amazon Echo, it is essential to monitor additional domains that are contacted by the Amazon Echo devices. For this, we introduce the *detection threshold D*. If an IoT service has N IoT-Specific domains, we require to observe traffic involving k IP/port combinations that are associated with $max(1, \lfloor D \times N \rfloor)$ of the N domains. To determine an appropriate value for this threshold, we rely on our ground truth dataset, see Section 5.4.

We start with 96 devices in our testbeds. We have multiple copies of the same device deployed on different continents. This reduces the set of devices to 56 unique products. Of these, many are from the same manufacturer, e.g., a Xiaomi rice cooker, a Xiaomi plug, and a Xiaomi light bulb. Since these devices are often supported by the same backend infrastructure of the manufacturer, the list of domains has significant overlap and often fully overlaps. In our methodology, we can detect 3 different IoT platforms, the coarsest level, as 4 of our products rely on them. Moreover, we generated rules for detecting 29 IoT devices at the manufacturer level. We had a diverse range of products from Amazon and Samsung in our testbed that allowed us an in-depth analysis and cross-examination of domains contacted by different products. Therefore, for devices using Alexa voice service (i.e. Alexa Enabled) and for Samsung IoT devices, we detect the former at the platform level and the latter at the manufacturer level. For Alexa Enabled and Samsung IoT devices, we compared the domains across different devices and obtained enough side information about the purpose of their domains that allowed us to further divide each of them into two subclasses at more fine-grained levels. For this, we defined a hierarchy under Alexa Enabled devices, namely Amazon products and Fire TV. Amazon products are detected at the manufacturer level and include products such as the Amazon Echo family and are the superclass of Fire TV. We identified 33 additional domains, besides the Alexa voice service domain, that were contacted by Amazon products. Moreover, Fire TV contacts up to 67 domains (34 more than Amazon products). This allows us to establish its subclass under Amazon products at the product level. Using side information [182] and comparing the set of domains across different Samsung products, we monitor 14

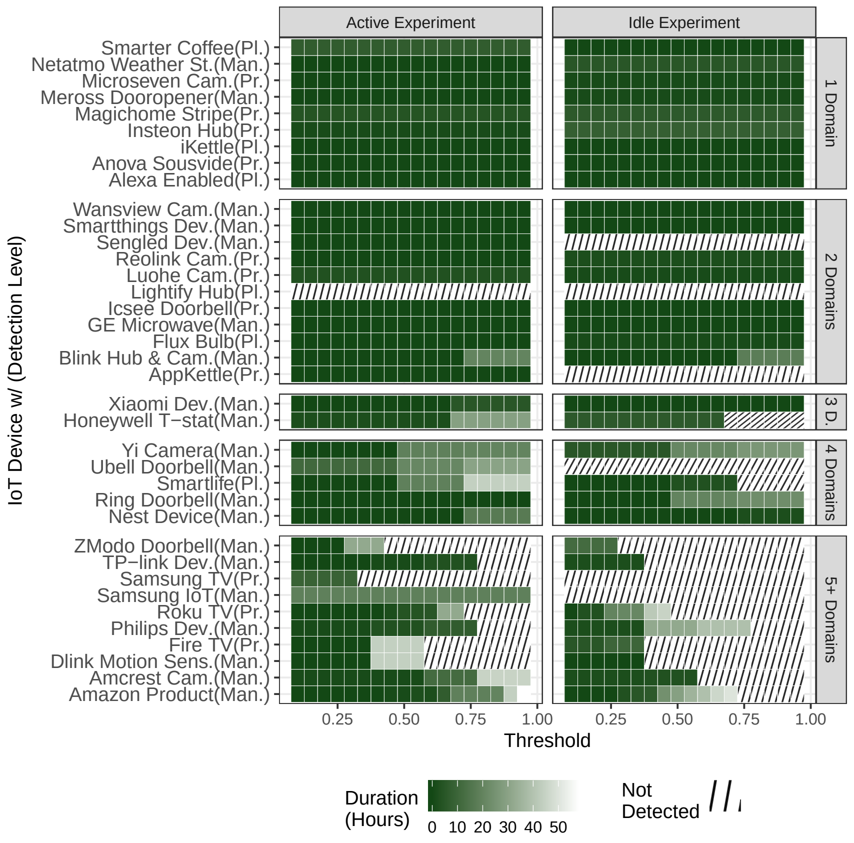


Figure 5.10: *Home-VP: Time to detect IoT (per threshold).*

domains in total, but only one domain is important to detect Samsung IoT devices with Samsung firmware (these include a broad range of products, such as fridges, washing machines, and TVs). Samsung TVs contact 16 additional domains not used by any other Samsung devices in our testbed.

Using the above methodology, except for the devices listed in section 5.3.2.3, we generated detection rules at different levels for our testbed devices. We generated rules for detecting 20 manufacturers and 11 products that amount to the 77% of manufacturers in our testbeds. We generate rules for 4 *unique* IoT platforms by monitoring 1 to 4 domains (4 devices contacted 2 platforms, and we report them separately). Finally, for 11 products, we consider between 1 to 67 domains; for a detailed number of domains per IoT device, see Figure 5.10.

5.4 Methodology: Crosscheck

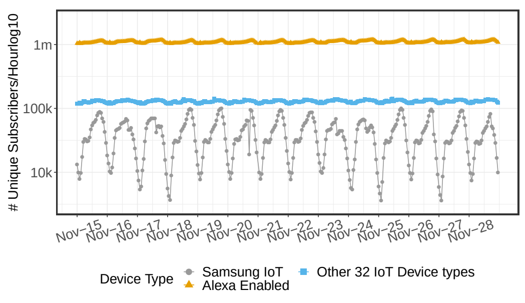
We use our ground truth dataset to check how long it takes for our methodology (applied to the sampled flow data from the ISP) to detect the presence of the IoT devices for the idle and the active experiments (see ④ of Figure 5.2). For this, we report the time that it takes to detect an IoT device that is hosted in our ground truth subscriber line when it is in active mode (Figure 5.10 left) and idle mode (Figure 5.10 right). We only include the ones that are detectable with our methodology, i.e. those that do not rely exclusively on shared infrastructures. We also annotate the device name with its detection levels: Platform (Pl.), Manufacturer (Man.), and Product level (Pr.).

On average, by requiring the evidence of at least 40% of domains, we are able to detect 72/93/96% of IoT devices that are detectable at the manufacturer or product level within 1/24/72 hours in the active mode. Even in idle mode, the percentage is 40/73/76% with 1/24/72 hours. For the devices detectable only at the product level (Pr.), with the same required evidence, we detected 63/81/90% of them within the 1/24/72 hours, respectively, in active mode. Note, we are using the sampled ISP data. Indeed, popular products such as Amazon products (i.e. Echo Dot, Echo Spot) can be almost instantly detected. This is a significant finding and underlines that it is possible to use sampled flow data within an ISP to accurately detect the presence of a specific IoT product within a subscriber line, despite differences in activity and IP churn due to operational requirements.

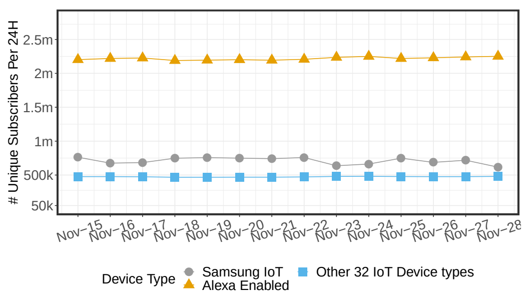
A closer look reveals that, in general, it takes longer to detect an idle IoT device in comparison to when it is active. This is not surprising, as most IoT devices show more network activity in active mode. However, this does not mean that the increase will occur across all of the services contacted by a device, since there are exceptions that take longer to detect, even in active mode, e.g., SmartLife, and Nest.

Figure 5.10 also contains information regarding the number of monitored domains per IoT device with their detection level. For 9 IoT devices, a single domain is considered. For the others, we consider many more (up to 67). A *threshold* determines the fraction of domains for which we require evidence of network traffic to claim detection. To understand the impact of such a threshold on detection time, we variate its value from 0.1 to 1 and show the corresponding detection times. Note, for IoT devices where we consider only one domain, the variation of the threshold does not change the detection time, as we always require evidence of at least one domain. Overall, we note that a larger threshold can increase the detection time, and some IoT devices may no longer be detectable. However, it may also increase the false positive rate. We crosscheck possible false positives by running another experiment where we only enable a small subset of IoT devices. We then apply our detection methodology to these traces and do not identify any devices that are not explicitly part of the experiment. We also try to avoid false positives by ensuring that the domain sets per device differ.

Regarding detectability, we noticed that 6 IoT devices could not be detected even after the entire duration of our idle experiments. A closer investigation showed that for 5 of these, the frequency of traffic is so small that their likelihood of detection is very low. Indeed, for this specific time period, they were invisible in the NetFlow data. This highlights that in order to be able to confidently detect a device, the device has to either exchange enough packets with the targeted domains or the sampling rate shall be increased. For Samsung TV, we require to observe enough domains to confirm the presence of a Samsung IoT device before moving forward with detection. Thus, if we do not see enough Samsung IoT domains, then we do not claim the detection



(a) *Per Hour.*



(b) *Per Day.*

Figure 5.11: *ISP: Per Hour, Subscriber lines with IoT activity (Alexa Enabled, Samsung IoT, and others).*

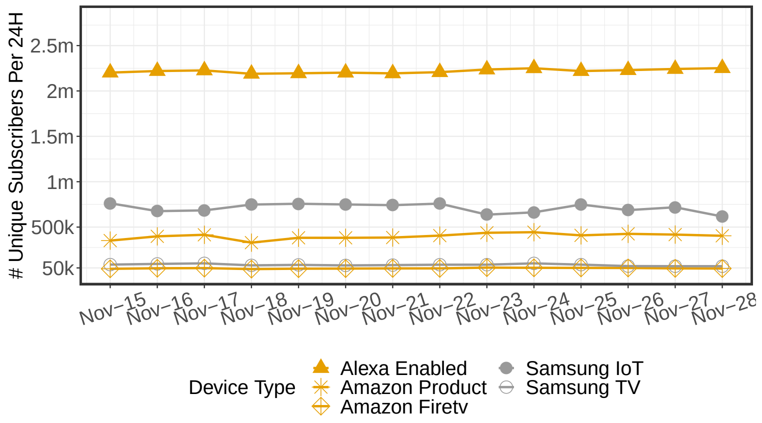


Figure 5.12: *ISP: Drill down for Amazon and Samsung IoT devices–per Day.*

of Samsung TVs. Nevertheless, the results look very promising for us to attempt to detect deployed IoT devices in the wild.

5.5 Results: IoT in the Wild

In this section, we apply our methodology for detecting IoT activity in the ISP and IXP data (see (5) in Figure 5.2). For this, we focus on the two weeks, in which, we collected the data from the ground truth experiments to obtain up-to-date mappings of domains to IPs.

5.5.1 Ethical Considerations and Privacy Implications

Applying our methodology to traffic data from ISPs and IXPs may raise ethical concerns as it may be considered as analyzing customer activities. However, this is not the goal of this study. The goal here is to showcase that it is possible to detect and map the penetration of IoT device usage. As such, this study is not about subscribers' device activities but about detection capabilities and aggregated

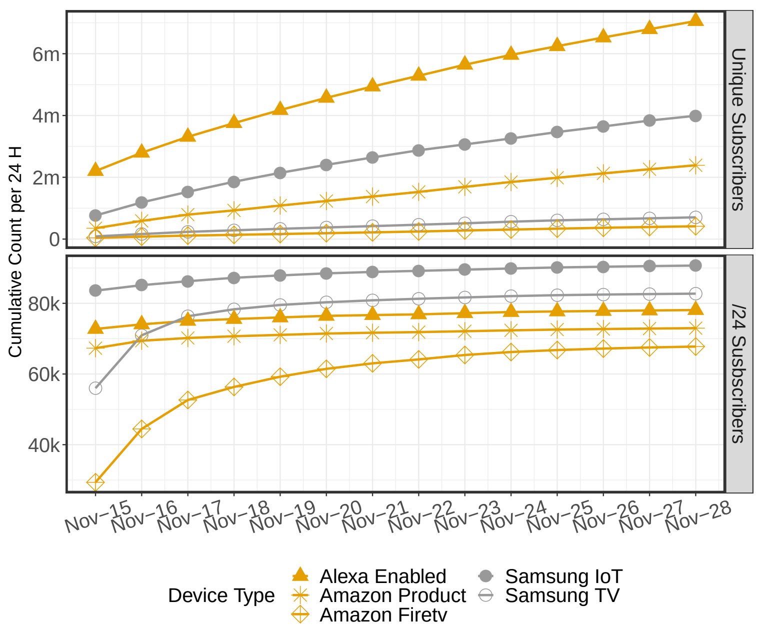


Figure 5.13: *ISP: Cumulative # of subscriber lines resp. /24s with daily IoT activity across two weeks.*

usage. Thus, we report on percentages of subscriber lines where we can observe IoT-related activity. Indeed, we cannot trace IoT activity back to individuals as the raw data was anonymized per recommendations by [170] and never left our collaborators' premises. Moreover, we do not analyze any data that is not related to the detection of IoT presence, e.g., DNS queries [39], or flows that are not related to IoT backend infrastructures, to eliminate any user Web visit profiling.

5.5.2 Vantage Point: ISP

IoT-related activity in-the-wild. Figure 5.11 shows the number of ISP subscriber lines for which we detect IoT-related activity. The ISP does not operate a carrier-grade NAT. Even if multiple IoT devices are hosted at an ISP subscriber, we count the hosting subscriber only once. Thus, the number of subscribers that host a given IoT device is a lower bound for the number of the given IoT device on the premises of ISP subscribers. Figure 5.11a and Figure 5.11b focus on hourly and daily summaries. Since the top IoT devices detected are Alexa Enabled and Samsung IoT, we show them separately. We see IoT-related activity for roughly 20% of the subscriber lines. Our results show a significant penetration of Alexa Enabled devices of roughly 14%. This is slightly more than estimates of national surveys in the country where the ISP operates, stating that the market penetration of Alexa Enabled devices, as of June 2019, is around 12% [183–185]. Yet, contrary to our study, these reports cannot capture which devices are in active use on any particular day, e.g., Nov. 2019. Note, in Figures 5.11, 5.12, 5.14 and 5.15 we apply our methodology on each time bin independently.

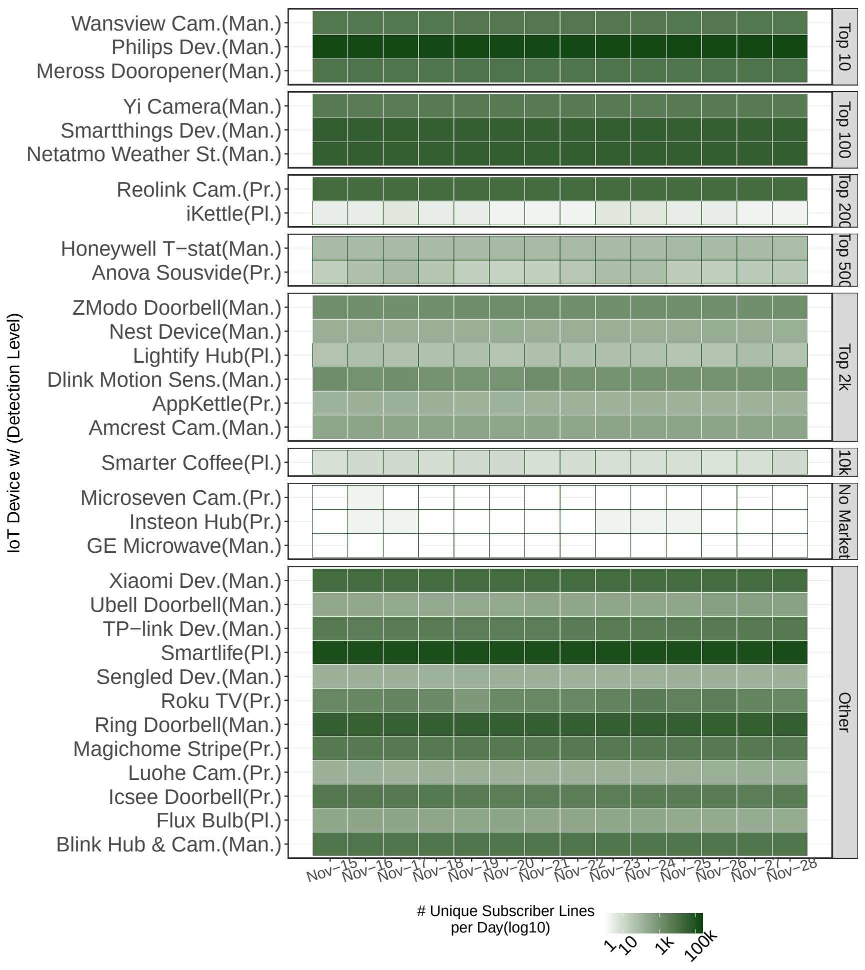


Figure 5.14: *ISP: Drill down of IoT activity for 32 different IoT device types with their popularity in the ISP's country.*

Daily patterns of IoT-related activity. By looking at the hourly plots in Figure 5.11a, we see some significant daily patterns for Alexa Enabled and Samsung IoT devices. We do not see diurnal patterns for the other 32 IoT device types. Such diurnal patterns are correlated with human activities. Typically, during the Day, network activity increases as the users interact with the IoT devices, while it decreases during the night when the devices are idle. As detection likelihood is correlated with network activity, the device's detectability also correlates with this diurnal pattern. We note that the patterns for Alexa Enabled do not differ from those for Samsung. The reason is that many of the Alexa Enabled and Samsung IoT (Samsung TVs) classes may be used more for entertainment, which is why their activity is higher in the evenings. Samsung IoT devices have a small spike in the mornings before gradually reaching their peak around 18:00 (ISP timezone).

For the drill down for Samsung IoT devices, see Figure 5.12. Even with a diurnal variation for Alexa Enabled, there is a significant baseline during the night. This is expected as IoT devices often have traffic, even when idle, and are thus detectable.

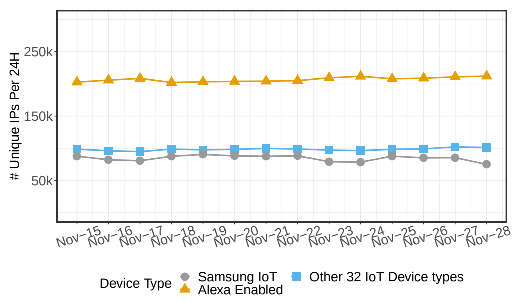


Figure 5.15: *IXP: Number of Samsung IoT, Alexa Enabled, and Other 32 IoT device types IPs observed/day.*

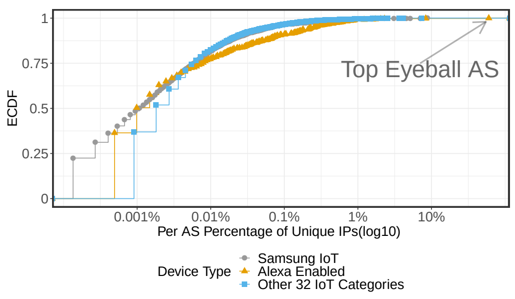


Figure 5.16: *IXP: ECDF of Per-ASN Percentage (# Unique IPs) - Day 15-11-2020.*

Over the course of a day, the diurnal variation is rather low compared with the typical network activity driven by human activity. This explains the low variance of the observed number of subscriber lines for Alexa Enabled devices.

Aggregation per day. We observed in Section 5.4 that, while it is often possible to detect Alexa Enabled devices within an hour, the same is not always true for Samsung IoT devices. Therefore, Figure 5.11b reports the same data but this time using an aggregation period of a day.[1] We see that the total number of observed subscriber lines does not change drastically from Day to Day. However, we also note that the number of subscriber lines with Alexa Enabled devices roughly doubled, while those with Samsung increased by a factor of 6. The reason is that detecting Samsung IoT devices is more challenging because they are contacting their Primary domain less frequently than Alexa Enabled devices. Thus, their detection is heavily helped by the increase in the observation time period. For the other IoT devices, we see these effects, whereby the increase correlates to the expected detection time. Note, both Samsung IoT and Non-IoT devices contact certain Samsung domains. In our analysis, we only consider domains that are exclusively contacted by *Samsung IoT* devices. By adding those domains, the number of detected Samsung devices will be increased at least by a factor of two, but this also adds false positives to our results.

Detecting specific devices. So far, we have focused on the superclass of Alexa Enabled and Samsung IoT devices. However, by adding more specialized features, our methodology allows us to differentiate them further. For example, some subsets of domains are only contacted by specific products. Thus, in Figure 5.12 we show which fraction of the Alexa Enabled IoT devices are confirmed Amazon products and which fraction of these are Fire TVs using a conservative detection threshold of 0.4. For Samsung IoT devices, we show how many of them are Samsung TVs. Again, the number of subscriber lines with such IoT devices is quite constant across days. As expected, the specialized devices only account for a fraction of the devices of both manufacturers.

[1]Most subscriber lines are not subject to new address assignments within a day. Most addresses remain stable as the ISP offers VoIP services.

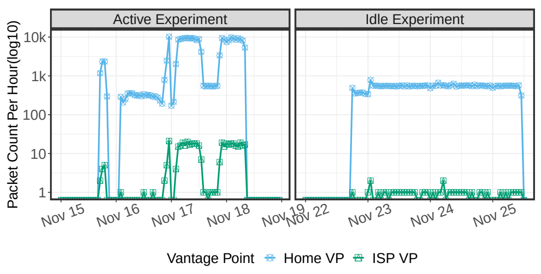


Figure 5.17: *Home-VP/GT Household: Single Alexa Enabled device.*

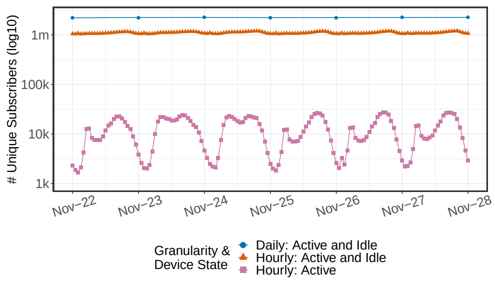


Figure 5.18: *ISP: # Subscribers with active Alexa Enabled/hour.*

Subscriber lines churn. While the ISP's overall churn of subscriber line identifiers is pretty low (as was also confirmed by the ISP operator), some changes are possible and may bias our results. Possible reasons for such changes are: unplugging/rebooting of the home router, regional outages, or daily re-assignment of IPs for privacy reasons. Yet, as most IoT devices are detectable within a day (recall Section 5.4), the churn should not bias our results. Still, to check for such artifacts, we move to larger time windows: see the upper panel of Figure 5.13, which plots the cumulative number of subscriber lines with detected Alexa Enabled and Samsung IoT devices, respectively, for up to two weeks. Here, we see that the fractions increase. However, we may have substantial double counting due to identifier rotation. To underline this conclusion, we consider penetration at the /24 prefix aggregation level; see the lower panel in Figure 5.13. The penetration lines stabilize smoothly, but at different levels and speeds. The latter is related to the popularity of an IoT device. If it is already popular, the likelihood of moving from a known to an unknown subscriber line identifier is lower with respect to less popular IoT devices.

Detecting other IoT devices in-the-wild. Figure 5.14 reports the detected number of IoT devices that are neither Alexa Enabled nor Samsung IoT. We report them using a heatmap, where each column corresponds to a day and each row to an IoT device annotated with its detection level. The color of each entry shows the number of subscribers lines during that Day. Our first observation is that the number of subscriber lines for each device class is very stable across the duration of our study. Next, we point out that our experiments include popular devices from both the European and the US markets. For reference, we report the relative popularity of each IoT device in the Amazon ranking for that device in the country where the ISP operates. If a ranking of a device is not available, we categorize them as "other." Popular devices are more prominent than unpopular ones or unavailable in the country's market. For example, on the one hand, there are Philips devices that are popular and in heavy use, with more than 100k daily subscription lines. On the other hand,, the Microseven camera is not in the country's market. Yet, we can still observe some deployments, and these results highlight that our methodology is able to detect both popular and unpopular IoT devices when the domains and associated service IPs that IoT devices visit can be extracted.

5.5.3 Vantage Point: IXP

Next, we apply our detection methodology at the IXP vantage point. Here, we have to tackle a few additional challenges: First, the sampling rate at the IXP is an order of magnitude lower than at the ISP. Second, the vantage point is in the middle of the network, which means that we have to deal with routing asymmetry and partial visibility of the routes. Third, while the ISP does aggressive spoofing prevention, e.g., with reverse path filtering, this is not possible at the IXP. Spoofing prevention is the responsibility of individual IXP members. Thus, we require TCP traffic to see at least one packet without flags, indicating that a TCP connection was successfully established. While this may reduce visibility, it prevents us from overestimating the presence of IoT traffic.

While the IXP offers network connectivity for every ASes, only a few member ASes are large eyeballs [186]. It is not surprising that we did not observe any activity of the ground truth experiment, recall Section 5.2. Still, we can detect significant IoT activity. Figure 5.15 shows the number of IPs for which we detected IoT activity per Day for our two-week study period (November 15th-28th, 2019). We are able to detect roughly 90k Samsung devices, 200k Alexa Enabled devices, and more than 100k other IoT devices. This underlines that our methodology, based on domains and generalized observations from a single subscriber line, is successful. Most IXP members are non-eyeball networks. As such, we expect that the detected IoT activity is concentrated on these members. Figure 5.16 shows an ECDF of the distribution of IoT activity per AS for one Day (November 15th, 2019) and three IoT device types, namely, Samsung IoT, Alexa Enabled, and the other IoT devices. The distributions are all skewed—a small number of member ASes are responsible for a large fraction of the IoT activity. Manual checks showed that these are all eyeball ASes. Yet, we also see a fairly long tail. This underlines that some IoT devices may not only be used at home (and, thus, send their traffic via a non-eyeball AS).

5.6 Discussion and Related Work

We previously covered the related work in IoT device detection in Section 3.3. In summary, multiple solutions use passively collected datasets to detect IoT devices. However, only a few studies apply their solutions *at scale* and report their findings. Authors in [38] proposed a method to identify IoT devices by observing passive DNS traffic and unique IP addresses that the device connects to. Unfortunately, many IoT devices rely on shared infrastructures, and often different IoT devices from the same vendor connect to the same servers; therefore, detection at the scale of ISP/IXP, based on the IP addresses and port numbers without considering the important role of shared infrastructures, cannot be very reliable. Following our studies, Authors in [34] used Machine Learning (ML) techniques to identify IoT devices in IPFIX data from a residential ISP to improve the detection accuracy. However, the sampling rate —if any— of their IPFIX data was unclear.

5.6.1 Device Usage Detection

A natural question is whether sampled flow data allows one to distinguish if an IoT device is in active use. Our results indicate that the answer is positive. First, our ground truth experiments show that for some devices, the domain sets used during the idle experiments differ from those during active experiments. Hence we can use these domains to determine an IoT device's mode (active/idle). Second, the amount of traffic also varies depending on the mode. To highlight this, Figure 5.17 shows the number of observed packets at the Home-VP for a single Alexa Enabled device, as well as the ISP-VP for both modes. Activities cause spikes above 1K at the home vantage points and above 10 at the ISP-VP. These ranges are never reached during the idle experiments.

When using the first insight for, e.g., devices from TP-link (TP-link Dev.), we are able to capture active use for only 3.5% of the devices. The reason is that these are plugs that have a total traffic volume so low that it limits the detectability due to the low sampling rate at the ISP. When using the second insight for Alexa Enabled devices, we find that we can detect significant activity. Figure 5.18 shows both the subscriber lines with Alexa-enabled devices per hour per day as well as the subscriber lines with active Alexa-enabled devices. Based on the observations mentioned above, we used the threshold of 10 for packet counts per hour to filter out subscribers that actively used Alexa-enabled devices in a given hour. Based on this threshold, we see that the number of actively used devices reaches 27,000 during the day and weekends (November 23rd-24th, 2019), following the diurnal pattern of human activity.

5.6.2 Potential Security Benefits

The ability to detect IoT services can be used constructively or even as a service by ISPs. For example, if there are known security problems with an IoT device, the ISP/IXP can block access to certain domains/IP ranges or redirect their traffic to benign servers. The methodology can also be used for troubleshooting, incident investigation, and even incident resolution. For example, an ISP can use our methodology for redirecting the IoT devices traffic to a new backend infrastructure that offers privacy notices or security patches for devices no longer supported by their manufacturers.

Moreover, if an IoT device misbehaves, e.g., if it is involved in network attacks or part of a botnet [187], our methodology can help the ISP/IXP in identifying what devices are common among the subscriber lines with suspicious traffic. Once identified, their owner can be notified in a similar manner, as suggested by [41], and it may be possible to block the attack or the botnet control traffic [188].

5.6.3 Limitations

Our methodology has some limitations.

Sample devices. We need to have sample devices in order to observe which domains are being contacted.

Superclass detection. We mostly check for false negatives and limitedly for false positives as we only have traffic samples from a subset of IoT devices, but not for all possible IoT devices. If an IoT device relies on a shared backend infrastructure or common IoT APIs, we only detect the superclass, e.g., at the manufacturer level.

Network activity. We rely on the network activity of IoT devices. As such, if the traffic volume is very low, detectability decreases, and detection time increases.

Shared infrastructures. We cannot detect IoT services that rely on shared infrastructures. If the IoT devices change their backend infrastructure, e.g., after an update, we may have to update our detection rules too.

5.6.4 Lessons Learned

Our analysis could be simplified if an ISP/IXP had access to all DNS queries and responses as they do in [38] and [39]. Even having a partial list, e.g., from the local DNS resolver of the ISP, could improve our methodology. Yet, this raises many privacy challenges. An increasing number of end-users rely on technologies like DNS over TLS [189], or public DNS resolvers, e.g., Google DNS, OpenDNS, or Cloudflare DNS, rather than the local ISP DNS server [190]. Yet, this also points to another potential privacy issue—the global data collection and analysis engines at these DNS operators, which can identify IoT devices at scale from the recorded DNS logs using our insights. Capturing DNS data from the network itself would require deep packet inspection and, thus, specialized packet capture, which is beyond the scope of this study.

The subscriber or device detection speed varies, depending not only on the device and its traffic intensity but also on the traffic capture sampling rates. The lower this rate, the more time it may take to detect a specific IoT device. Moreover, identifying the relevant domains for each IoT device does require sanitization, which may involve manual work, e.g., studying manuals, device documentation, vendor websites, or even programming APIs. Given that we are unable to identify IoT services if they are using shared infrastructures (e.g., CDNs), this also points out a good way to hide IoT services.

5.7 Chapter Summary

Home IoT devices are already popular, and their usage is expected to grow further. Thus, we need to track their deployment without deep packet inspection or active measurements, both intrusive and unscalable methods for large deployments. From this chapter, our insight is that many IoT devices contact a small number of domains, and, thus, it is possible to detect such devices at scale from sampled network flow measurements in very large networks, even when they are in idle mode. We showed that our method could detect millions of such devices in a large ISP and in an IXP that connects hundreds of networks.

Our technique was able to detect 4 IoT platforms, 20 manufacturers, and 11 products–both popular and less popular ones–at the vendor level and in many cases even at product granularity. In addition, we showed that 20% of 15 million subscriber lines of the ISP used at least one of the 56. While this detection may help understand the penetration of IoT devices at home, it raises concerns about the general detectability of such devices and the corresponding human activity.

6 IoT Devices: A Case Study on Leaking Users' Privacy

In the previous chapter, we developed a methodology to detect IoT devices in an ISP. One security and privacy benefit of detecting IoT devices was that ISPs could investigate which devices are common among subscribers with suspicious traffic. This motivated us to use our methodology in a specific case study. In this chapter, we study how deployed devices at homes can affect users' privacy. We collaborated with the large European ISP and focused on IPv6 subscribers. We identified a privacy leakage caused by devices using legacy IPv6 addressing mechanisms and measured how many subscribers were potentially affected by this leakage. While conducting this study, we also put our IoT detection technique into use.

The adoption of IPv6 on the Internet is continuously increasing [191]. One of the drivers is the unprecedented demand for smart devices at home, ranging from voice assistants to smart TVs and surveillance cameras, that all have to be assigned addresses to access the Internet and the cloud [192]. While the use of Network Address Translation (NAT) and concerns about IPv6 addressing privacy have delayed its adoption, operators, vendors, and the research community have long ago provided privacy solutions to mitigate these risks. ISPs have adopted prefix rotation [193] and network equipment manufacturers and software developers have enabled IPv6 privacy extensions [194, 195].

A recent work [196] shows that if the home network gateway router, also referred to as customer premises equipment (CPE), is using a legacy IPv6 addressing standard employing EUI-64 (Extended Unique Identifier), it is possible to track devices that use IPv6 at home using active measurements. Unfortunately, in this chapter, we report that even if the CPE and the ISP apply the best common practices, i.e. IPv6 privacy extensions and prefix rotation, it is still possible to track devices that use IPv6 at home. In detail, we show that the existence of only a single device that uses EUI-64 at home can spoil the privacy of potentially all IPv6-enabled devices and eventually end-users' privacy across these devices. Third-party providers, such as hypergiants [197], network traffic aggregators (Internet exchange point, upstream providers), or service providers (e.g., NTP, DNS providers), receiving connections from devices at the same home can potentially defeat the privacy of current IPv6 solutions even if only one of these devices uses the legacy EUI-64 technique. Unfortunately, the average end-user is not in a position to know which of their devices use EUI-64.

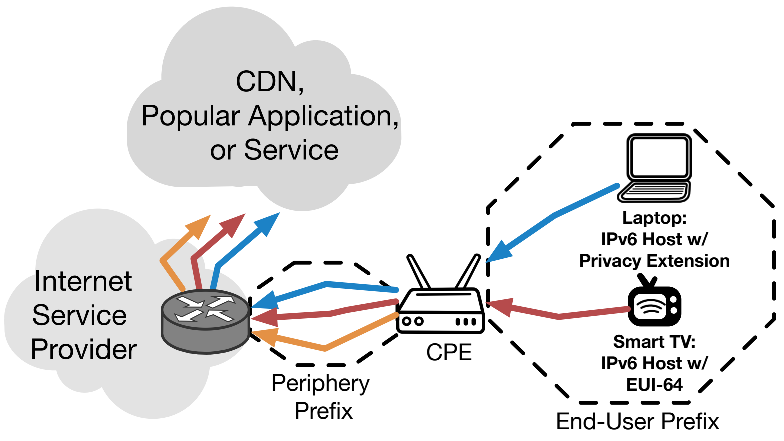
6.1 Background

To solve the address shortage in IPv4, among other things, the networking community introduced the IPv6 protocol more than two decades ago [198, 199]. Nevertheless, IPv6 is only recently being deployed on a larger scale [200] with about 36% of all requests to Google going over IPv6 as of March 2022 [201]. In addition to the IPv6 address space being larger, the addressing itself is also different compared to IPv4 [202]. While in IPv4, most end-user clients get their address via DHCP [203], in IPv6 clients get addresses either via DHCPv6 [193] or stateless address auto-configuration (SLAAC) [204]. Instead of directly assigning a full address as in DHCP or DHCPv6, with SLAAC, a router sends a prefix to its clients (i.e. the network part), and the clients, then by themselves, choose an IPv6 address within that prefix (i.e. the host part). This host part is called interface identifier or *IID*. Initially, the IID part used an encoding of the interface's MAC address, called *EUI-64* [205]. The unique and consistent nature of MAC addresses lead to devices being trackable over time and across different networks [206]. Consequently, IPv6 *privacy extensions* were proposed, which randomize the IID part instead of using a device's MAC address [194]. In addition to user devices being trackable by EUI-64 addresses, ISP subscribers can also be tracked by their prefix. In order to defeat prefix tracking, ISPs can change the prefix of each customer after a certain time (*prefix rotation*). Although there has been several works on IPv6 measurements [206–229], many of them focused on active measurements or structural properties of the IPv6 space. Rye *et al.* recently published the work closest to ours citerye2021follow, in which they show that prefix rotation can be defeated by tracerouting customer premise equipment (CPE), which responds with EUI-64 addresses. In our work, we show the privacy implications of EUI-64 usage among devices directly within the end-user network.

6.2 Methodology

In this section, we describe our methodology and show how a single device using EUI-64, i.e. not using privacy extensions, can be used to track devices at the subscriber level. In Figure 6.1, we show how an end-user prefix can be tracked despite the ISP performing frequent prefix rotation. In the example scenario, there are two devices in the end-user prefix, a laptop, and a smart TV. Both are using IPv6, the former with privacy extensions, the latter with EUI-64. The CPE device also has IPv6 connectivity on the upstream facing interface. If the CPE device's WAN-facing address is not within the end-user prefix, it can not be used for tracking with our methodology.

Since the smart TV is not using privacy extensions, it allows CDNs and other large players on the Internet to track not only the smart TV itself but all devices within that end-user prefix. In fact, we can use the smart TV's IID part of the IPv6 address as its unique tracking ID since it is derived from a MAC address. Furthermore, we assign this same *tracking ID* to all addresses within the end-user prefix. This



Time	Host	IPv6 Address	Tracking ID
1	SmartTV	2001:db80:1111:b000:8e8f:90ff:fe12:3456	8e8f:90ff:fe12:3456
1	Laptop	2001:db80:1111:b000:ddde:abcd:1111:ff11	8e8f:90ff:fe12:3456
1	CPE	2001:db80:2222:b25d:aff0:abff:fe34:5679	aff0:abff:fe34:5679
...			
2	SmartTV	2001:db80:3333:fff1:8e8f:90ff:fe12:3456	8e8f:90ff:fe12:3456
2	Laptop	2001:db80:3333:fff1:1111:1123:ee11:2222	8e8f:90ff:fe12:3456
2	CPE	2001:db80:2222:b266:aff0:abff:fe34:5679	aff0:abff:fe34:5679

Figure 6.1: *Privacy leakage across prefixes.*

way, we can jointly track all devices of a subscriber by relying on a single EUI-64-enabled device. After the initial blue and red flows were observed, the ISP rotates the customer's prefix (time 2), and all customer devices are now using a new IPv6 address. Importantly, as the smart TV is still using the same IID even in this new prefix, any content provider can again associate all devices with the same tracking ID as before. With this technique, a single EUI-64 device in an end-user subnet can spoil the privacy gains of prefix rotation of all other devices, even if they use privacy extensions.

For our method to be effective, the devices in the same end-user prefix must contact a vantage point. In our case, we are in a privileged position and see all connections and thus, can track all the devices. However, in the wild, these devices would require to contact the same application, e.g., hypergiants, content delivery networks, search engines, upstream providers, or other popular services such as DNS or NTP. The devices can then simply be tracked by assigning tracking IDs to the red and blue flows as shown in Figure 6.1.

Recall that the IID part of a EUI-64 IPv6 address is generated by inserting the `ff:fe' hex string between the third and fourth bytes of a MAC address and setting the Universal/Local bit. We can extract the MAC address from the EUI-64 part of an IPv6 address and uncover the device manufacturer. To achieve this, we extract the Organization Unique Identifier (OUI) part of the MAC address, i.e. the first three bytes. For the mapping, we use the official IEEE OUI database [230]. This database contains information about the name and address of the manufacturer that registered the OUI.

6.3 Datasets

ISP Profile: We analyze data from a large European Internet Service Provider (ISP) that offers Internet connectivity to more than 15 million broadband subscriber lines in Europe.

IPv6 Assignment at the ISP: The ISP fully supports IPv6 by utilizing dual-stack addressing. Each CPE device gets delegated a /56 IPv6 prefix, out of which it will pick one /64 prefix, which is then used to assign addresses to clients via SLAAC.

By default, the ISP rotates the /56 prefixes delegated to customers every 24 hours. Generally, the IPv6 prefix used for the upstream-facing CPE interface to the ISP ("periphery prefix" in Figure 6.1) may or may not share the same prefix as the end-user network. Thus, in the latter case, a /56 prefix that does not contain an upstream-facing CPE interface represents an end-user network. We will show in our analysis in Section 6.4.2 that the CPE interface and end-user networks of this ISP do not share the same /56 prefixes.

ISP Data: The data is sampled network flow data collected at the ISP using NetFlow [50] to assess the state and operation of its network routinely, a typical operation of ISPs. For our analysis, we apply our method to the NetFlow data at the premises of the ISP, and we do not transfer or have direct access to the NetFlow data. The data was collected in July 14, 2021, and four months later, on November 17, 2021.

6.4 Privacy Violations at the Edge

To assess the prevalence of privacy violations due to devices without privacy extensions, we apply our methodology on NetFlow data of the ISP (see the previous section). Since the ISP rotates the customer prefixes once a day, we analyze one day of data, namely, Wednesday, July 14, 2021, to show the feasibility of tracking devices at home. We also examine the data collected on Wednesday, November 17, 2021, which confirms our initial observations. Unless otherwise mentioned, our results refer to the first dataset.

6.4.1 Quantifying EUI-64 Prevalence

In Figure 6.2 (left), we report the number of IPv6 addresses visible in the ISP during one day. Recall that the ISP serves around 15 million subscriber lines. The number of non-EUI-64 addresses—in our case, those are IPv6 addresses with privacy extensions enabled (see Section 6.4.6 for a detailed analysis of non-EUI-64 addresses)—is more than 100 million. This is to be expected as these devices frequently use new IPv6

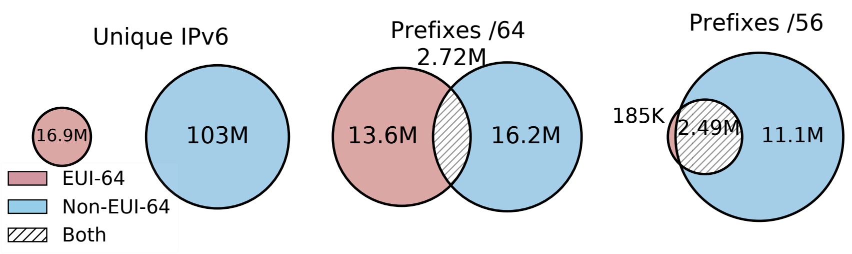


Figure 6.2: *Venn diagram for EUI-64 and non-EUI-64 IPv6 addresses and the overlap between different prefix sizes.*

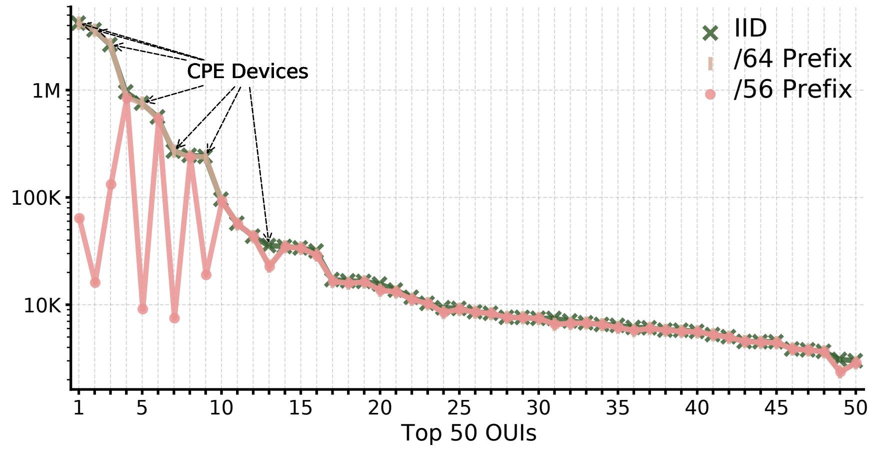


Figure 6.3: *OUI popularity. Note that the y-axis is log-scaled.*

addresses, and more than one of these devices may be served by a subscriber line. On the other hand, the number of IPv6 addresses for devices that do not use privacy extensions, i.e., EUI-64, is smaller, around 17 million. However, we have strong and consistent identifiers for IPv6 addresses used by these devices, i.e., their IIDs, that we use to track devices even when the ISP performs prefix rotation. In total, we found 14.4 million devices that use EUI-64.

Next, we map all IPv6 addresses to their /64 prefix. We see that the numbers are now quite similar, 13.6M for EUI-64 and 16.2M for prefixes with non-EUI-64 addresses, with an overlap of 2.7M prefixes.

As mentioned in Section 6.3, the ISP assigns /56 addresses to each subscriber line. In Figure 6.2 (right), we illustrate the number of prefixes that contain devices that use EUI-64, non-EUI-64 devices that use privacy extensions, and the prefixes that contain both types of addresses(i.e., dual-type prefixes). In total, we observed at least one EUI-64 device in around 2.68 million /56 prefixes out of 11.3 million /56 prefixes. Thus, the number of affected /56 prefixes accounts for about 22.2%. Note that the vast majority (more than 93%) of the host prefixes with EUI-64 devices also host non-EUI-64 devices as well. This shows that the presence of privacy-violating EUI-64 addresses impacts a substantial portion of ISP subscribers. Even within a day, it is still possible for prefix rotation to happen for some subscriber lines. We can detect these rotations for EUI-64 using prefixes by tracking the IIDs across multiple /56 prefixes. We observed that only less than 13% of the EUI-64 using /56 prefixes

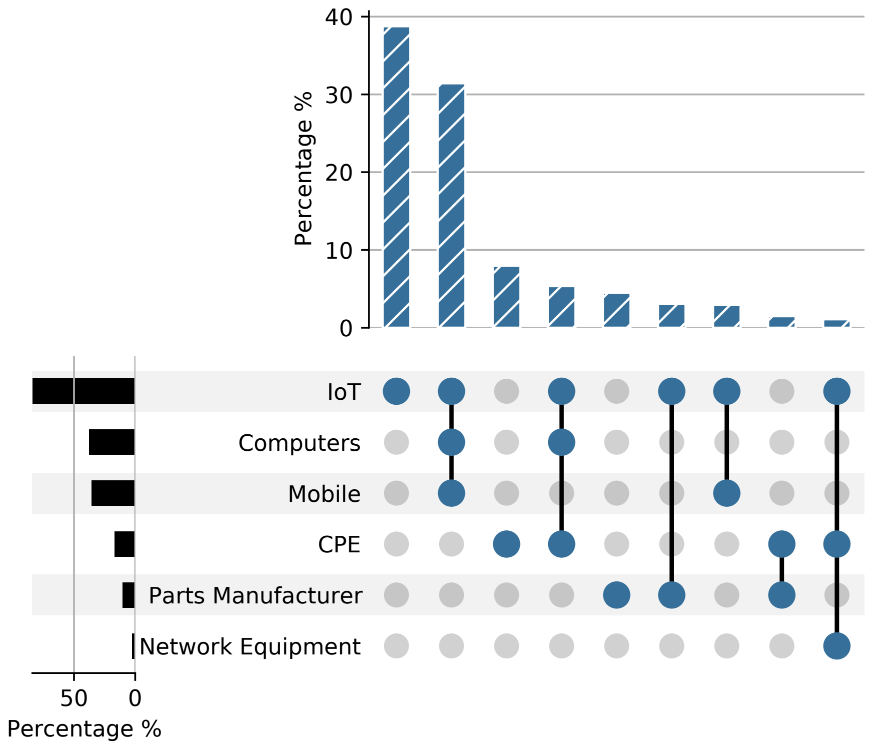


Figure 6.4: *EUI-64 addresses mapped to different device categories. The percentage corresponds to the number of /56 prefixes.*

had prefix rotation within a day. Hence, if the same IID is observed across multiple /56 prefixes, we count the prefixes only once. For non-EUI-64 prefixes, we cannot track them across prefixes after prefix rotation, which is precisely the purpose of using IPv6 privacy extensions.

6.4.2 Popularity of EUI-64 Manufacturers

Based on the IPv6 address for devices that use EUI-64 addresses, we analyze the device manufacturer using the OUI (see Section 6.2). In total, we find devices with 1216 unique OUIs from 1113 distinct manufacturers. In Figure 6.3, we show manufacturers sorted by popularity (i.e. number of unique IIDs). We focus on the top 50 manufacturers as these are responsible for more than 99.1% of all IIDs. A closer investigation shows that 6 out of the top 10 are CPE manufacturers. The rest in the top 10 are IoT, smart TV, mobile devices, home appliances, and data storage manufacturers.

Interestingly, the number of covered /56 prefixes or even /64 ones is almost identical to the number of IIDs in most cases. This means that it is expected to be one device from each manufacturer in each /56 or /64 in our dataset. A striking difference is the case of CPEs, where the number of /56 prefixes is substantially lower than the /64 prefixes and the corresponding IIDs. We attribute this to two reasons. First, the

WAN (upstream-facing) interfaces of the CPEs in the ISP typically do not share the same /56 prefix as the devices at home, i.e., the periphery prefix is different from the end-user prefix shown in Figure 6.1. We confirm this with multiple users of the ISP. Second, the IPv6 address of the WAN interface of CPEs are concentrated within a relatively small number of prefixes that it seems the ISP uses for exactly this purpose. Thus, in our methodology, the IPv6 address of the CPE is not sufficient to track the devices at home. On the other hand, it is possible to use this information to defeat the privacy of devices at home with active measurements [196].

Based on these insights, we re-estimate the number of affected prefixes by differentiating between periphery subnets used by CPEs and end-user subnets used by devices at home. We find that around 2.23M prefixes out of a total of 2.6M EUI-64 prefixes are end-user prefixes. Therefore, about 19% of all 11.3M /56 prefixes are at risk of privacy leakage.

6.4.3 EUI-64 Manufacturer Categorization

To understand what type of devices contribute the most to the leakage of users' privacy due to EUI-64, we characterize the business model and products of the associated manufacturers. We associate our OUIs to their manufacturers using The IEEE OUI database [230], which contains details such as the name and address of the company that registers an OUI. Depending on the manufacturer's range and type of products, it is possible to even determine a device's type. For example, if we observe an OUI registered by a company producing only wind turbines, the device generating the traffic is likely a wind turbine. To this end, we manually visit the website of the top 100 manufacturers found by our method. We categorize their products into one or multiple categories. We consider the following business types and any combinations: IoT, computers, mobile devices, CPEs, part manufacturers, and network equipment manufacturers (see Table 6.1 for a detailed description of each category). Some companies produce generic products, e.g., network interface cards (NICs) installed on many devices. In such cases, we mark their OUIs as Parts Manufacturers. Moreover, if a company produces more than one product category, we assign its OUIs to all those categories. Finally, we assign a weight to each manufacturer with the associated coverage of /56 prefixes. Then, we aggregate the weights for the manufacturers of the same type.

As shown in Figure 6.4, around 39% of the prefixes that host EUI-64 devices contain products from manufacturers that only produce IoT devices. The second most popular category, which accounts for 32%, are devices by manufacturers active in different product lines that include IoT devices, computers, and mobile devices. All other categories account for 8% or less. Thus, the large majority of subscribers with EUI-64 devices are IoTs or likely IoTs. To our surprise, a large number of subscribers host computers, mobile phones, and other equipment that also uses EUI-64. Although large vendors, e.g., Apple, by default enable privacy extensions in their products [231], other popular vendors do not. This could be related to some operating systems not enabling privacy extensions by default.

Category	Description
IoT	Manufacturers of internet-connected devices such as sensors, smart TVs, home appliances, security cameras, alarms, smart speakers, etc.
Computers	Laptops, personal computers, and servers
Mobile	Mobile phones and tables
CPE	Devices supporting broadband technologies such DSL, cable modem, and 5G/4G hotspots.
Parts Manufacturer	Network interface cards, CPUs, memory modules, motherboards, WiFi modules, and chipsets that can be embedded into other devices.
Network Equipment	Routers, switches, access points, and firewalls.
Gaming Console	Internet-connected devices primarily used for gaming.
Unknown	Manufacturers that we were not able to find their website or were not providing any information about the type of their products.
Virtual Machine	Vendors that develop virtual machine and hypervisor software.

Table 6.1: *Description of device categories.*

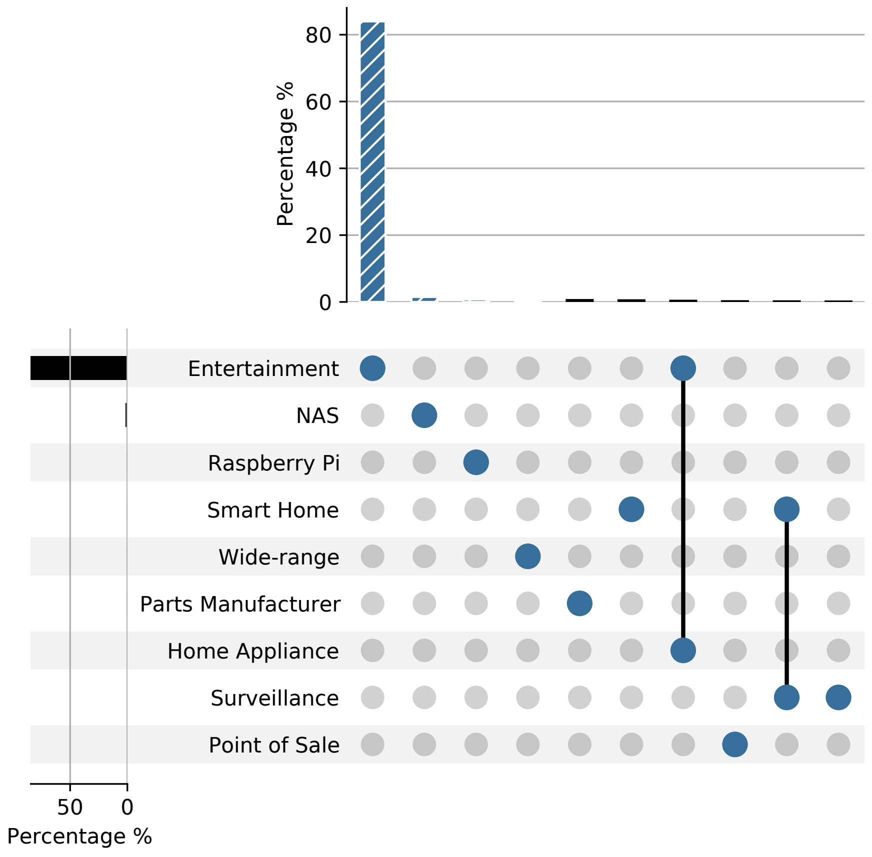


Figure 6.5: *Composition of IoT-only manufacturers. The percentage corresponds to the number of /56 prefixes.*

Manufacturer Type	Description
Entertainment	Manufacturer of smart TVs, over-the-top streaming devices (OTTs), smart speakers, and network-connected media players.
Network Attached Storage	Internet-connected devices used for storing data.
Smart Home	devices such as smart plugs, light bulbs, door openers, alarms, and thermostats.
Varied	Manufacturers with a large portfolio of IoT devices that not only includes all our categories but spans beyond them. For example, robots, industrial devices, highly-specialized medical equipment, etc.
Parts Manufacturer	Chipsets, and modules tailored to be used specifically in IoT devices, e.g., 3G/4G and Zigbee modules. Note, we tag a manufacturer in this category only if it explicitly states that it produces IoT-specific modules and chipsets.
Home Appliance	Washing machine, refrigerators, air conditioners, air purifiers, etc.
Surveillance	Security cameras and related surveillance equipment.
Point of Sale	Devices mostly used at retail stores for accepting payments.

Table 6.2: *Description of IoT manufacturer categories.*

6.4.4 EUI-64 Use Among IoT Devices

Next, we focus on the IoT devices that contribute the most to the leakage of users' privacy. We take a conservative approach by only considering manufacturers which exclusively produce IoT devices. We manually investigate their product line and further categorize their products as follows: entertainment (that includes a smart TV, voice assistants, streaming devices, media players), network attached storage (NAS), Raspberry Pi, smart home equipment, IoT parts manufacturer, home appliances, surveillance devices, point of sale devices, and varied products (that include multiple categories). See Table 6.2 for a detailed description of these categories. As Figure 6.5 shows, the most popular category is entertainment IoT devices, which cover more than 85% of all /56 prefixes with only IoT devices. In this category, we identify more than 19 popular manufacturers. If this relatively small number of manufacturers had adopted best common practices to enhance IPv6 privacy, EUI-64 privacy leaks could have been substantially reduced.

However, even at the level of a manufacturer, it is possible that different products or product versions have different behavior when it comes to privacy leakage. To assess how common this is, we consider our dataset's top contributor of EUI-64 IoT devices ("manufacturer 1"). Using the methodology that we introduced and validated in our previous chapter (See Chapter 5), we annotate the products of this IoT manufacturer based on the contacted destination addresses. We utilize the destination information to annotate the most popular IoT product of this manufacturer ("product A"), and we also infer if a specific device uses EUI-64 or not, based on the IPv6 address. In Figure 6.6, we show the cumulative unique number of IPv6 addresses for all the products with EUI-64 of the manufacturer and the number of devices with product A with and without EUI-64 with hourly updates. A first observation is that, as

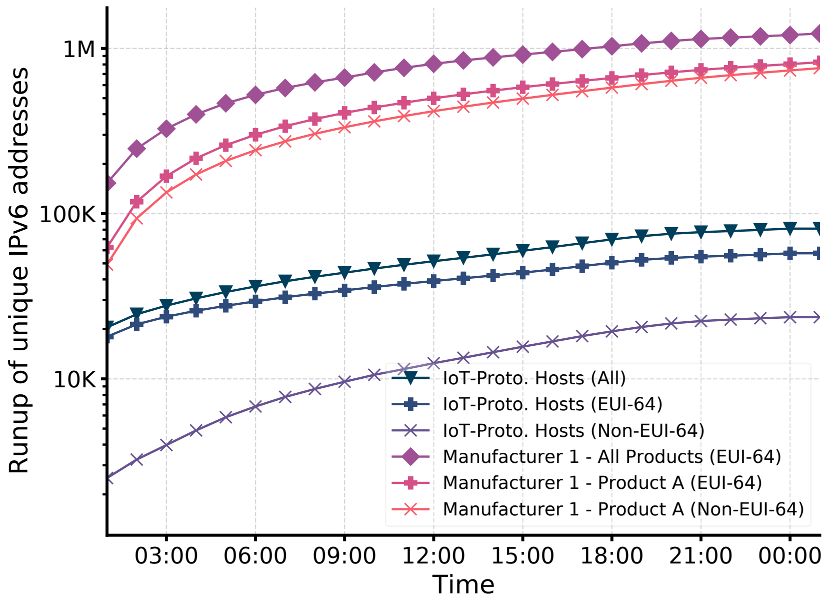


Figure 6.6: *Prevalence of EUI-64 in IoT devices and a case study for a popular IoT product. Note that the y-axis is log-scaled.*

expected, within 24 hours, the number of IPv6 addresses that host the EUI-64 devices, as well as the IIDs of this manufacturer, converge to around 1.2 million and 650k IPv6 addresses for the total and product A, respectively. The number of IPv6 addresses and IIDs that do use and do not use EUI-64 is similar for product A. Even though some of the devices belonging to product A have adopted IPv6 privacy extension, either by updates or because of newer models, the majority of these devices still have the potential to leak user privacy.

Unfortunately, it is not easy to generate signatures for all IoTs based on the visited destinations because the IoT devices have to be purchased, and communication data has to be collected in a lab over longer periods of time [232]. On the other hand, IoT-specific protocols such as MQTT [233] are popular among many IoT manufacturers. Indeed, we notice that port TCP/8883, i.e., the IANA-assigned port for MQTT, is among the top 10 ports by our top manufacturers (see Figure 6.7 for a detailed view of ports used by different manufacturers). Hence, we use this activity as a proxy to infer what is the percentage of IoT devices that use EUI-64 vs. any other MQTT activity that does not use EUI-64. In addition, we confirm that more than 95% of these devices contact servers exclusively used for IoT cloud services [14, 15]. Therefore, these devices are highly likely to be IoTs. Our analysis in Figure 6.6 shows that more than 83% of the devices that communicate using the common IoT protocol MQTT are also using EUI-64. This is another indicator of the rampant privacy-violating practice of using EUI-64 addresses among IoT devices.

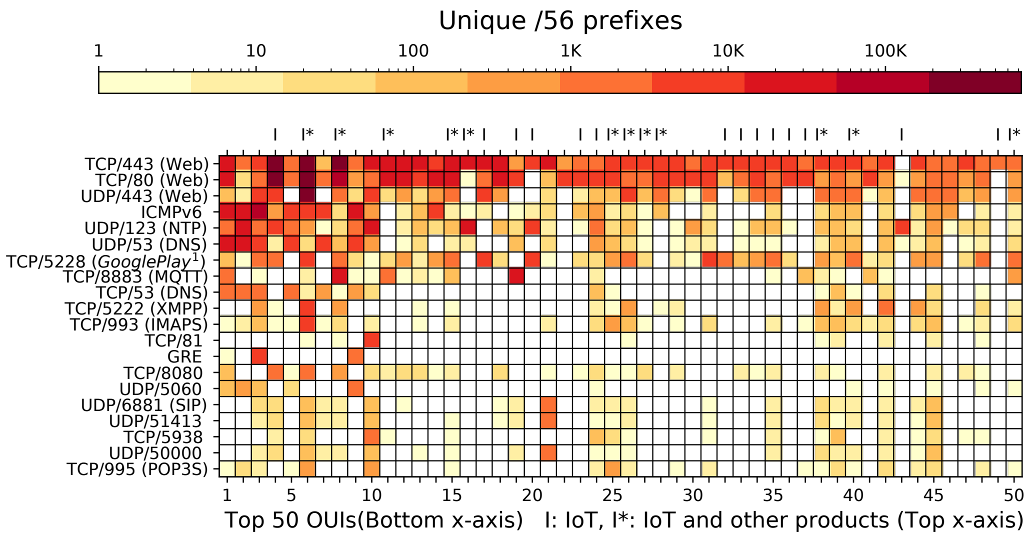


Figure 6.7: *Heatmap showing used application ports for the top 50 OUIs. [1]The Google Play port is not officially registered with IANA.*

6.4.5 Traffic Profile by Manufacturer

Figure 6.7 shows the popularity among the top 20 protocols utilized by the top 50 manufacturer devices that use EUI-64. These devices utilize protocols that are also popular for other devices, like laptops, smartphones, etc., that may use privacy extensions. Thus, it is possible that devices using EUI-64 and ones that do not use EUI-64 contact the same CDNs (Web on ports 80 and 443), applications (Google play updates on port 5228 [234], MQTT on port 8883), or other services (NTP on port 123, DNS on port 53).

6.4.6 Analysis of Non-EUI-64 IPv6 Addresses

Non-EUI-64 addresses can be privacy extension addresses, addresses assigned via DHCPv6, or also statically assigned addresses. In order to understand how many of the non-EUI-64 addresses are indeed privacy extensions addresses, we analyze the interface identifier (IID) of all non-EUI-64 addresses. We use the Hamming weight, i.e. the number of bits set to '1', to analyze the random nature of IIDs. In completely random 64-bit IIDs, i.e. the presence of privacy extensions, we would expect exactly half of the bits to be set to '1'. Moreover, the central limit theorem states that the sum of those independent IID Hamming weight distributions tends toward a normal distribution. In Figure 6.8 we show the Hamming weight distribution of these IIDs, and the normal distribution shifted one bit to the left due to the universal/local bit. As can be seen, the non-EUI-64 Hamming weight distribution perfectly matches the normal distribution. Consequently, non-EUI-64 addresses in our dataset are, in fact, privacy extension addresses.

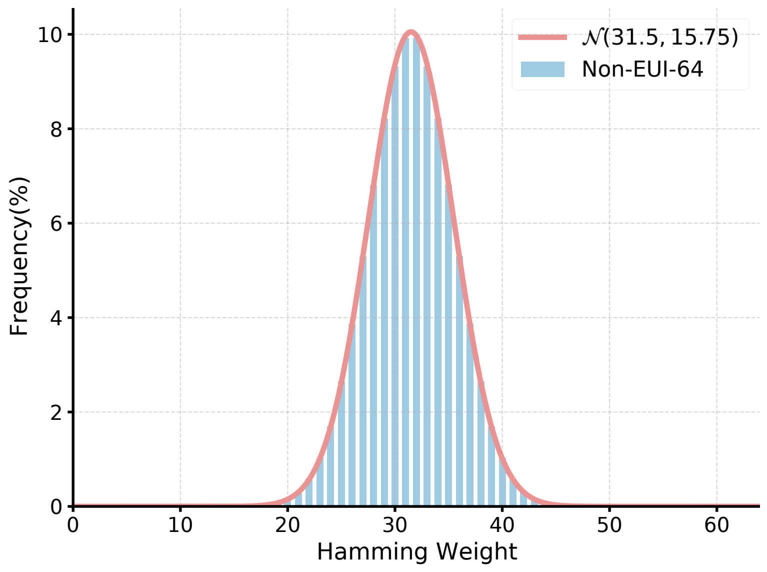


Figure 6.8: *Hamming weight distribution of non-EUI-64 IIDs.*

6.4.7 Collateral Privacy Leakage

In this section, we turn our attention to the popularity of EUI-64 devices in end-user prefixes. As shown in Figure 6.9, we typically only find one or two EUI-64 devices per end-user prefix. Indeed, more than 90% of end-user prefixes that host both EUI-64 and non-EUI-64 devices, i.e., are dual-type prefixes, have two or fewer EUI-64 devices. Only about 1% of dual-type prefixes host more than five EUI-64 devices. Recall, from Figure 6.2, more than 93% of all end-user prefixes with EUI-64 devices also host non-EUI-64 devices.

Also, in Figure 6.9, we see that number of non-EUI-64 addresses in dual-type prefixes is larger than the number of EUI-64 addresses. However, a single EUI-64 device is sufficient to leak user privacy to a third party if both this device and a non-EUI-64 device contact the same destination. To understand how probable this collateral privacy leakage is, first, we analyze the popular applications that are contacted by EUI-64 devices. Our analysis shows that these devices contact popular applications, e.g., Web (port 443, 80), DNS (port 53), NTP (port 123). For details about the popularity of ports for the top EUI-64 manufacturers, we refer to Figure 6.7. This is alarming, as other devices that use IPv6 privacy extensions also contact these ports. To estimate the collateral damage, we count the number of dual-type prefixes where EUI-64 and non-EUI-64 devices contact the same third-party provider. Figure 6.10 shows the number of end-user dual-type prefixes which can be tracked over time by common hypergiants [235]. We find that in total, two million end-user prefixes (around 17% of the total end-user prefixes) are affected by this collateral privacy leakage, with the top hypergiants, i.e., HG1, HG2, and HG3, being able to longitudinally de-anonymize prefix rotation efforts by the ISP. Alarmingly, users do not even need to log in or visit the websites of these hypergiants to be tracked. Tracking can simply happen by accessing one of their services, e.g., loading ads or static files. Some of

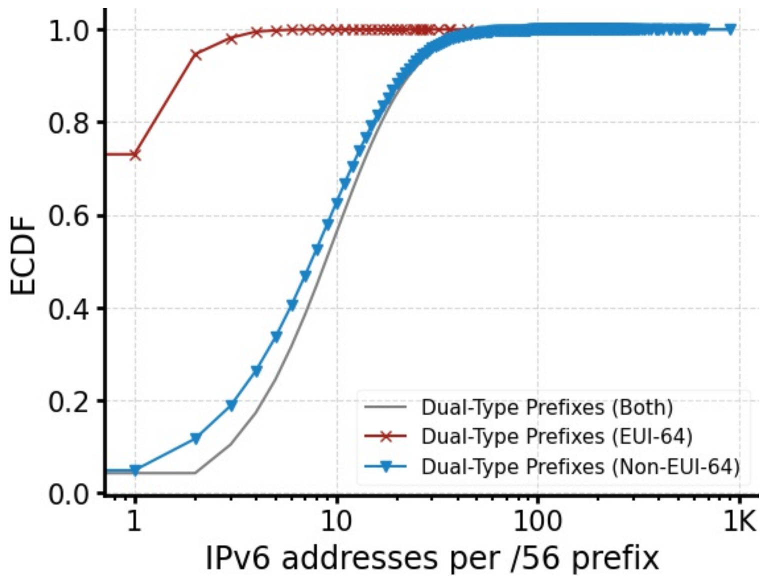


Figure 6.9: *IPs in prefixes with both EUI-64 and non-EUI-64 IPs, i.e. dual-type prefixes. The number of non-EUI-64, EUI-64, and both types of IPs in dual-type prefixes. The X-axis is log-scaled.*

these hypergiants run popular public DNS services and online advertising platforms, making them attractive as a destination. A recent study also shows that services such as NTP can collect a vast number of IPv6 addresses [229], thus, breaking IPv6 privacy when sufficient conditions are in place, as we describe in our methodology (cf. Section 6.2). We note that this form of tracking can not only be facilitated by hypergiants but also at major aggregation points in the network, such as peering locations, Internet exchange points, transit providers, and large data centers.

6.5 Discussion

Vendor Self-regulation: Hardware vendors should adequately test their products and make every effort to protect the privacy of their consumers, as currently, there is a gap in legislation regarding IPv6 privacy. This includes all parties involved, from chip manufacturers to product integrators, software companies, and ISPs. For software companies, e.g., operating system distributors, it is important to enable IPv6 privacy extensions by default. Unfortunately, many Linux distributions do not activate privacy extensions by default at the time of writing. Products using Linux derivatives in their software likely unknowingly risk their users' privacy. This could be related to the fact that the original privacy extensions specification [194] contained a recommendation to deactivate them by default. The current standard [195] does not contain this recommendation anymore. We, therefore, recommend that all IPv6-capable software stacks enable IPv6 privacy extensions by default. We are in contact with hardware vendors to make them aware of this issue.

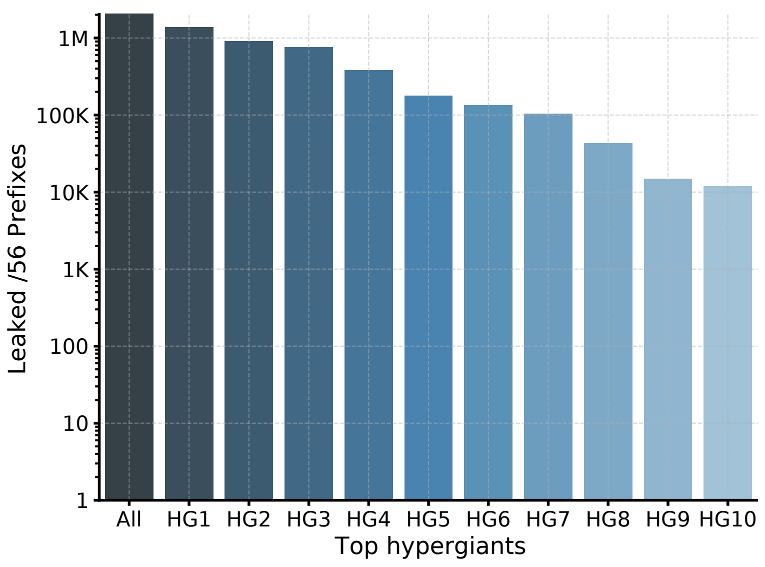


Figure 6.10: *Per hypergiant number of /56 prefixes vulnerable to privacy. Each of these prefixes contains at least one EUI-64 address and one non-EUI-64 address.*

Privacy Badges: The average user is not a privacy expert when purchasing or operating smart home appliances or other Internet-connected devices. Although the end-user may be aware of privacy risks when using such devices, we can not expect end-users to perform experiments to validate which devices use privacy extensions and which do not. The consumer unions and regulators, e.g., the FCC and FTC in the US and the European Commission in the EU, could require vendors to certify their products for IPv6 privacy compliance. These badges could affirm a product's compliance with the relevant future legislation, similar to other certifications, e.g., health, safety, and environmental protection standards [236].

The Role of the ISP: ISPs should continuously improve the privacy that they provide to their customers and could also inform them about potentially privacy-risky products in the market and their home network upon customer request. Another possibility would be to introduce a NAT in ISP IPv6 client networks. This would, however, break the end-to-end principle—a primary design goal of IPv6 [237]. Therefore, we refrain from recommending NAT as a practical workaround. Finally, ISPs should also check CPEs for privacy risks before shipping them massively to their customers.

6.6 Chapter Summary

In this chapter, we investigated how millions of deployed devices can affect the privacy of a large number of users at an ISP. We conducted a case study on IPv6 subscribers and showed a new way to defeat IPv6 privacy even when the ISP does prefix rotation and all devices except one at subscriber's private network use IPv6 privacy extensions.

We showed that even if all devices in an end-user prefix use privacy extensions, one single device with EUI-64 is enough to be used as an anchor to track them. Our analysis revealed that up to 19% of our subscribers could face this privacy leakage. Third parties such as hypergiants, network traffic aggregators (Internet exchange point, upstream providers), or service providers (e.g., NTP, DNS providers) receiving connections from devices at the same home can potentially track up to 17% of our subscribers. We showed that IoT devices are the largest contributors to this privacy leakage and, to a lesser extent, personal computers and mobile devices. We proposed that vendors self-regulate and enable privacy extensions and that regulatory intervention is necessary to protect users' privacy.

7 IoT Backend Servers: A Deep Dive into Backend Providers

In previous chapters, we focused on the IoT devices component of the IoT ecosystem. In this chapter, we focus on the IoT backend servers. Specifically, we characterize the infrastructure and traffic patterns of top IoT backend providers. These companies sell IoT backend server infrastructure and related services to the users. We characterize their deployment strategies, security incidents and traffic patterns observed in our ISP.

Although IoT devices offer increasingly complex and reach applications, many can not independently execute all parts of these applications. Many IoT devices lack the required compute, memory, and energy resources for computationally demanding applications or additional data; thus, it is common to offload part of the application to a backend in the "cloud". For example, applications that rely on machine learning are often easier to operate in the cloud, which is computationally more powerful and has readily available machine learning libraries [238, 239], rather than on the IoT device itself. A low-cost IoT camera typically streams its video to the cloud, where the main computation takes place, e.g., to identify suspicious activity and trigger an alarm in real-time. Moreover, many companies that use IoT devices commercially, e.g., within a production line or for logistics, collect all data in the cloud for analytics and operational decisions [240]. Thus, these clouds act as the *backend* of IoT applications.

IoT devices also lack the storage required, e.g., for content-centric applications. Thus, such IoT devices need IoT backend servers to download or upload content required by the application. For example, content recommendations require the user's profile and merge it against the available content [241], which may not be possible on the IoT device itself. Moreover, IoT device security and functionality often depend on an IoT backend. One prominent example is software updates—many IoT devices periodically check if software updates are available. Other IoT vendors or application providers push notifications to the IoTs when such updates are available.

As the number of deployed IoTs and their functionality increases rapidly, the demands for the IoT backend–in terms of capabilities and traffic–also increase. During the last years, we have observed a shift toward building special-purpose clouds to support IoT applications and cope with the increasing demand. Recently, the big technology giants, such as Amazon [181], Google [242], and Microsoft [16] started to offer IoT

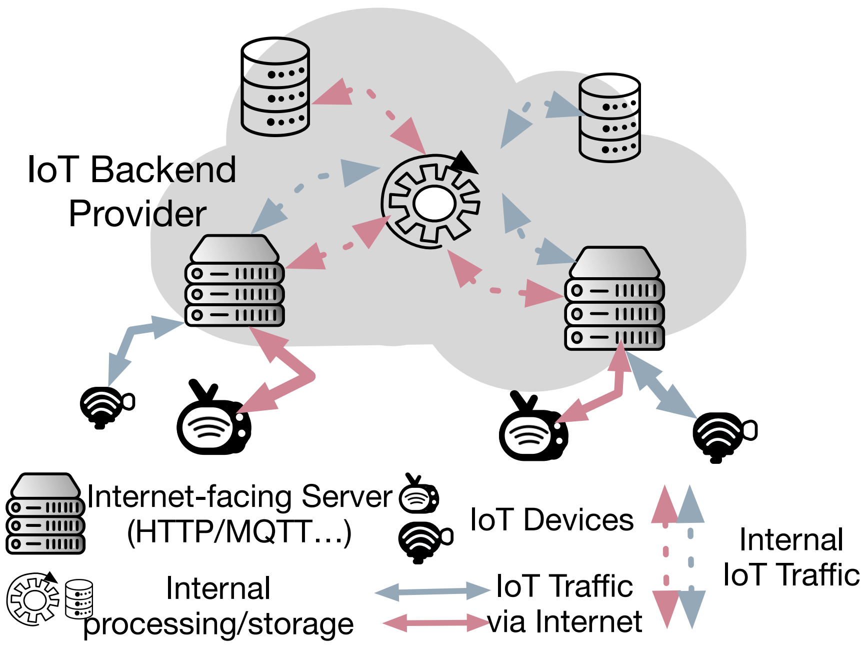


Figure 7.1: *IoT backend provider architecture.*

backend solutions as-a-service. Such companies are IoT backend providers and enable third-party IoT application providers to scale up and deliver their solutions to potentially billions of IoTs deployed around the globe.

Despite the critical role that these IoT backend providers play in the operation and security of IoT applications [11, 40, 137, 243], little is known about their locations, strategies, and volume share. Indeed, much of the work in the IoT area has focused on the inference of IoT clients [27, 39, 232] or general-purpose cloud providers or content delivery networks [197, 244, 245] that may also support IoT services. In this chapter, we turn our attention to the IoT backend providers. We develop new methods to identify their footprints and gain insights into their modus operandi.

Scope of the study: Our study was curiosity-driven, and we tried to understand the evolving IoT backend ecosystem to inform future studies by computer scientists, economists, and policymakers. As we are not aware of the companies' business strategies, we do not take a position regarding their deployment decisions and operation. Rather, we characterize the current state of the IoT ecosystem. This study was not a head-to-head comparison of different and possibly competing IoT companies.

7.1 Scenario and Related Work

In this section, we first describe the setting of our measurement study and introduce terminology. Then, we summarize related work.

7.1.1 IoT Backend Providers

Today, many of the IoT vendors [246–249], technology giants [242, 250–254], and cloud providers [16, 181] offer sophisticated IoT platform solutions. These solutions allow developers to deploy new services, support existing applications, collect data, or remotely manage and configure IoT devices. Typically, these IoT platforms have three major components: (*i*) software/hardware on the IoT device, (*ii*) Internet-facing gateway servers, and (*iii*) the internal storage/processing systems, e.g., for machine learning. Figure 7.1 depict the main components of a generic IoT platform. In this chapter, we identify and characterize the public part of IoT platforms, i.e., the Internet-facing gateways, that we refer to as *IoT backend*, see Figure 7.1. IoT backends facilitate data exchange between IoT devices and internal systems. We refer to companies that operate such infrastructures as *IoT backend providers*.

Note that IoT platforms are sophisticated entities. Thus, the focus of our study is their publicly announced IPs—the gateways between the IoT devices and the internal systems of the IoT platforms. Thus, the following aspects are out of scope: (*i*) the software/hardware installed on the IoT devices, (*ii*) internal processing/storage systems of IoT platforms, in particular, since these are typically not publicly accessible, and (*iii*) private interconnections between cloud providers and IoT platforms [255].

7.1.2 Related Work

In Chapter 3, we provided an overview of studies on IoT backend servers. We highlighted the need for further studies investigating the IoT backend servers. To this end, we are aware of a small number of studies that focus on IoT backend providers, whereby their main focus is also on security. Alrawi et al. [40] perform a security evaluation of home-based IoT deployments and highlight the need to understand IoT platforms, i.e., IoT backend providers. He *et al.* [137] develop fingerprinting techniques to classify traffic exchanged with the cloud as IoT-related or non-IoT-related traffic. A study by Zhou et al. [243] investigates five popular IoT platforms that enable smart home IoT applications. The study shows that these platforms are vulnerable to a number of remote attacks, including device substitution, device hijacking, device denial of service, illegal device occupation, and firmware theft. Jia et al. [11] report on the vulnerabilities of defense mechanisms used by popular IoT platforms for IoT-specific protocols, e.g., MQTT. We note that none of the prior works characterizes IoT backend provider footprints or their traffic flows.

7.2 Methodology

In this section, we discuss how we use a diverse set of sources, including documentation by IoT providers, active and passive DNS measurements, and IPv4/IPv6 scans to identify the set of backend IPs of each IoT backend provider, see Figure 7.2.

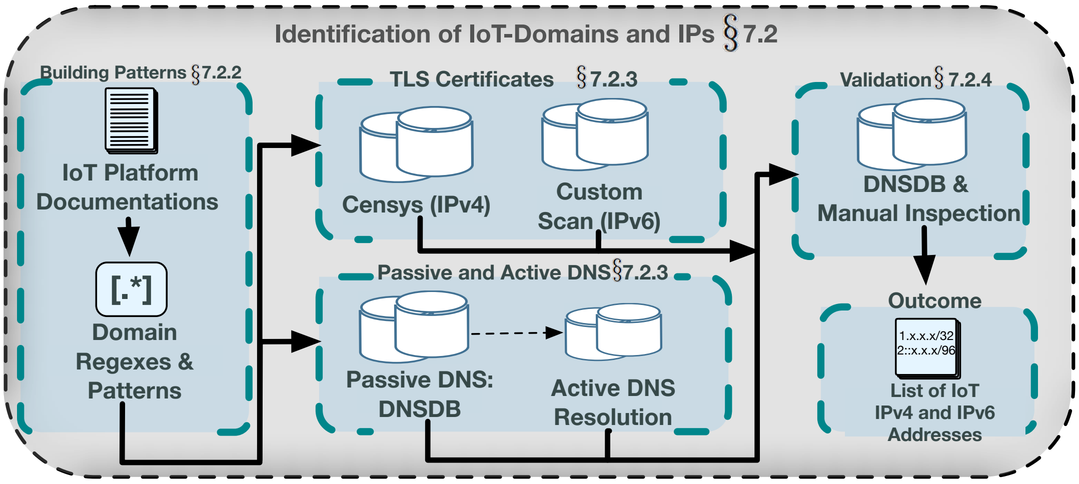

Identification of IoT-Domains and IPs §7.2
Building Patterns §7.2.2
IoT Platform Documentations
[.*]
Domain Regexes & Patterns
TLS Certificates §7.2.3
Censys (IPv4)
Custom Scan (IPv6)
Passive and Active DNS §7.2.3
Passive DNS: DNSDB
Active DNS Resolution
Validation § 7.2.4
DNSDB & Manual Inspection
Outcome
1.x.x.x/32
2::x.x.x/96
List of IoT IPv4 and IPv6 Addresses

Figure 7.2: *Our methodology to infer IoT backends' footprint.*

7.2.1 Selection of IoT Backend Providers and Study Periods

To compile a list of IoT backend providers, we consider the popular ones that were mentioned in previous studies [11]. We also expand this list by considering IoT backends operated by other major manufacturers, e.g., Cisco, Huawei, and Siemens, and cloud providers, e.g., Oracle, Tencent, and SAP. The complete list of the IoT backend providers in our study is presented in Table 7.2. By some accounts, for example, the IoT platform market research report by IoT analytics [256], these IoT backend providers are in the top 17 in terms of estimated revenue and are responsible for more than 90% of the total revenue.[1] We focus on the week starting from February 28 at midnight and ending on March 7, 2022. We also collect preliminary results (only IPv4) for December 3 to 10, 2021. Since the results are consistent, we focus on the week starting in February for all but Section 7.5.1.

7.2.2 Identification of IoT Domain Patterns

For each IoT backend provider, we start by identifying the domain names and the IP prefixes that are used for IoT backends. This information is often contained in the publicly available documentation since IoT vendors and device programmers need it. When a backend provider explicitly discloses their IPs, we use them for validation (see Section 7.2.4). Typically, IoT backends domains follow a well-defined form `<subdomain>.<region>.<secondlevel-domain>`, where:

- `<subdomain>` is either a domain of specific IoT services or a unique identifier (e.g., a hash or the name of an IoT platform customer). Some companies such as Alibaba, Tencent, and Bosch, also list the network protocol, e.g., MQTT, CoAP, etc. [246, 257, 258].

[1]Note that these reports are not peer-reviewed and we use their reported IoT backend revenue only for a rough estimation of their market share. We use neither the ranking nor or the revenue of these companies in our methodology.

- `<region>` indicates the full name or code of a city, a country, a region, or a continent;
- `<second-level-domain>` is either the second-level domain name of a parent company of the IoT backend provider or a special domain name allocated for the IoT backend.

However, some providers, e.g., Google, use the same fully qualified domain names (FQDNs) for all of their customers. In such cases, we use these FQDNs. In Table 7.2 we report the 16 IoT platforms for which we were able to generate regular expressions for their IoT backend domain names using the official documentation.

We leverage the structure of the IoT backend domain names to generate the regular expressions. If the <subdomain> part of the IoT domain is a unique value, e.g., a hash or a random string, we replace it with a regex wildcard. Similarly, we replace the <region> part of the IoT domains with appropriate regex terms that match the naming scheme of the different regions of the provider. Note that we also obtain the naming schemes for the regions of the providers from their documentation. Finally, we concatenate the regex terms with the <second-level-domain> to create the regular expressions. See Table 7.1 for examples.

7.2.3 Identification of Server IPs

Next, we use the above regular expressions to identify the IPs of possible IoT backend servers. Hereby, we rely on two complementary techniques. First, we take advantage of the information available in TLS certificates. Second, we use passive DNS data, namely, DNSDB [172, 175]. Finally, we complement the data with an additional active DNS dataset.

TLS Certificates.

Censys [21] continuously scan the IPv4 address space. In addition to scanning for open ports across a wide range of port numbers, it performs protocol-specific handshakes to collect banners and provides metadata, e.g., geolocation; these results are published daily. Motivated by the previous results [197], we use daily snapshots matching our study period to identify certificates with domains that match our regular expressions. The corresponding IPs are IoT backend provider IPs. Note, we only use certificates [259, 260] that is valid during the study period.

During our study period, Censys scans only include IPv4 addresses. To identify IPv6 addresses, we run active measurements using various IPv6 hitlists [208]. Our hitlists include IPv6 addresses that showed activity for popular IoT ports, i.e., 443 (HTTPS), 8883 (MQTT), 1883 (MQTT), and 5671 (AMQP). We add support for these IoT protocols to ZGrab2 [261] and we use it to collect TLS certificates from these IPv6 addresses. We perform this data collection from a server located in Europe. For a discussion on ethical considerations, we refer to Section 3.6.1.

DNS. We complement the above data with DNS data because scanning services typically only download the default certificates. In some cases, scanning services may

Provider Name	Data Source	Api Type	Regular Expression/Query
Huawei	DNSDB	Flexible Search	.\.(iot-(coaps|mqtts|https|amqps|api|da)\.).+\.myhuaweicloud\.com\.$/A
Amazon	DNSDB	Flexible Search	(.+)(\.iot\.)([[:alnum:]]+(-[[:alnum:]]+)+)?(\.amazonaws\.com\.$)/A
Oracle	DNSDB	Flexible Search	(.+\.|∧)(iot\.)([[:alnum:]]+(-[[:alnum:]]+)*\.)?(oraclecloud\.com\.$)/A
Baidu	DNSDB	Flexible Search	.\.(iot\.)([[:alnum:]]+(-[[:alnum:]]+)*\.)?(baidubce\.com\.$)/A
Siemens	DNSDB	Flexible Search	.(\.eu1\.mindsphere\.io\.$)/A
Sierra Wireless	DNSDB	Flexible Search	(.+\.|∧)(na\.airvantage\.net\.$)/A
Bosch	DNSDB	Flexible Search	(.+\.|∧)(bosch-iot-hub.com\.$)/A
IBM	DNSDB	Flexible Search	(.+\.|∧)(internetofthings\.ibmcloud.com\.$)/A
Microsoft	DNSDB	Flexible Search	(.+\.|∧)(azure-devices\.net\.$)/A
Tencent	DNSDB	Flexible Search	(.+\.|∧)(tencentdevices\.com\.$)/A
Tencent	DNSDB	Basic Search	rrset/name/*.tencentdevices.com./A
Google	DNSDB	Basic Search	rrset/name/mqtt.googleapis.com/A
Cisco	DNSDB	Basic Search	rrset/name/*.ciscokinetic.io./A
Amazon	Censys	String Search	*.iot.us-east-2.amazonaws.com
Amazon	Censys	String Search	*.iot.us-east-1.amazonaws.com
Amazon	Censys	String Search	*.iot.us-west-1.amazonaws.com
Amazon	Censys	String Search	*.iot.us-west-2.amazonaws.com
Huawei	Censys	String Search	*.iot-mqtts.cn-north-4.myhuaweicloud.com
Alibaba	Censys	String Search	*.iot-amqp.cn-shanghai.aliyuncs.com
Alibaba	Censys	String Search	*.iot-as-http.cn-shanghai.aliyuncs.com
SAP	Censys	String Search	*.iot.sap

Table 7.1: *An excerpt, less than %5, of regular expressions and queries for a subset of IoT Backend providers.*

not even be able to download the certificates, i.e., if the IoT backend provider (e.g., Google) requires to supply the domain name via Server Name Indication (SNI) header. In addition, other IoT backend providers such as Amazon, require the installation of a *client certificate*, in particular, for IoT protocols. In the absence of this certificate, the TLS handshake will fail.

DNS is another source of data for mapping domain names to IPs. DNSDB is a passive DNS database that contains historical DNS queries and replies for both IPv4 and IPv6 from multiple resolvers around the globe. We choose DNSDB as it supports regular expressions and time-range queries. For each IoT platform, we use DNSDB to collect all IPv4/IPv6 addresses in the response for queries where (*i*) the domain name matches the regular expressions for the IoT platform, and (*ii*) the query was issued within our study period. In addition, during our study period, we also performed daily active DNS resolutions for all domains identified via DNSDB. To perform these resolutions, we used three locations: two in Europe and one in the United States, see Section 3.6.2 for ethical considerations. Compared to a single location, using three vantage points increases our IP address coverage by $\approx 17\%$.

7.2.4 Validation of Server IPs

At this point, we have identified IPs related to IoT backend services. However, we do not know if they are used exclusively for IoT services or if they also host other services, e.g., Web services. In addition, we validate the accuracy and coverage of discovered IP addresses against ground truth for three IoT backend providers.

Shared vs. dedicated IPs: To identify IP addresses in our candidate sets that also provide services unrelated to IoT, we use a methodology similar to the one by Saidi et al. [232] and Iordanou et al. [174]. For each candidate IP, we use DNSDB to identify all the domain names that resolve to that particular IP. Next, we count the number of domains that do not match the IoT domain pattern yet, map to the IP. If this count exceeds a threshold, we assume it is not exclusively used to offer IoT backend services. Through this process, we detect IoT backend providers using CDNs or hosting non-IoT services. While choosing the threshold, for example, we discover that Google uses two sets of IPs: one exclusively for IoT MQTT traffic and another for HTTPS traffic that is also used for other Google services. In our IoT traffic flow analysis (Section 7.4) we focus only on those parts of the infrastructure that are exclusively used for IoT.

Validation against ground truth: While not all IoT backend providers publicly share their used IP ranges, three do this at least partially. Our methodology identified all the publicly listed IP addresses for Cisco and Siemens. Microsoft lists network prefixes for its IoT backend service, which correspond to more than 12,000 IP addresses. Using our methodology, we identify 484 of these IPs. All of them are within the listed prefixes. We conduct a study using traffic data from a large European ISP, see Section 7.4, and check the traffic to the listed prefixes. We only identify 52 active IPs. Out of these, our methodology misses only 4 IPs, leading to an underestimation of the IoT traffic volume of less than 1%.

7.2.5 Contribution of Each Dataset

Using as baseline the data collected on the Feb. 28, 2022, in Figure 7.3 we show the contribution of each data source grouped per IoT provider. The plot includes both IPv4 and IPv6 backends—the bar for IPv6 is shaded. We distinguish IPs extracted from “TLS Certificates” (discovered via Censys), our IPv6 scans, “Passive DNS” (discovered via DNSDB), “Active DNS” (identified via our active resolutions), and “Multiple Sources” (addresses discovered by at least two methods).

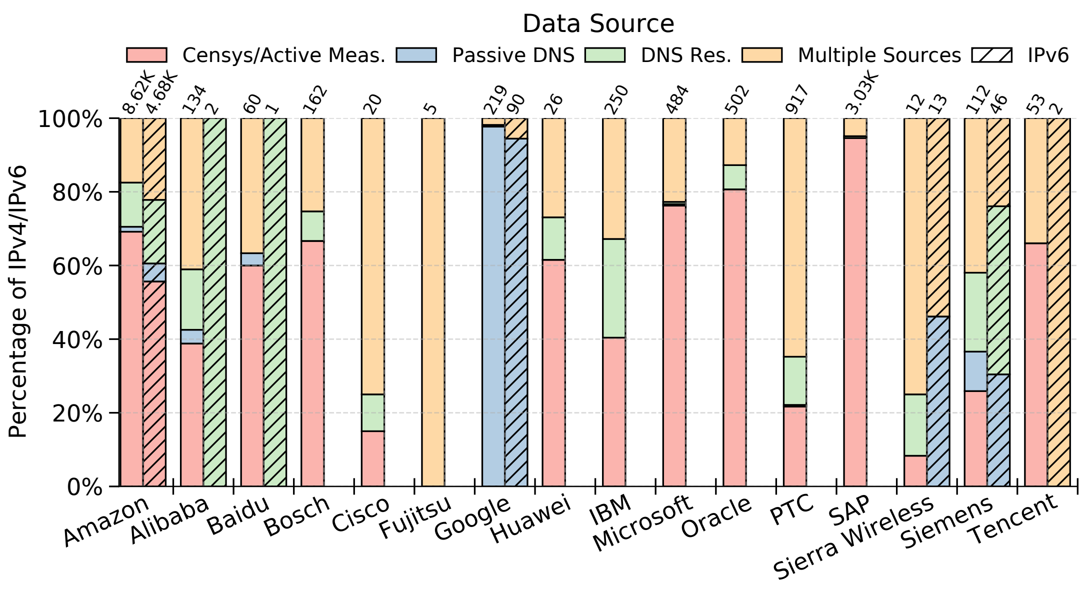


Figure 7.3: *Fraction and # of IPs per provider per source (left bar IPv4, right bar IPv6).*

First, we notice that some of the IoT backend providers only support IPv4 addresses. Second, there is no consistency regarding a preferred data source. For example, when using only Censys data, we detect all IPs of the IoT backends for Microsoft, SAP, and Tencent. However, we identify less than 2% of the Google IPs. The reason for this is that Google is using TLS SNI. Thus, a majority of Google's IoT platform IPs are discovered using passive DNS. The contribution of passive DNS is also significant (more than 5%) for Siemens, Alibaba, and Sierra Wireless (IPv6). Our active DNS resolution is able to discover close to 20% of Alibaba (IPv4), Amazon AWS, Huawei, Bosch, Cisco, IBM, PTC, Siemens, and Sierra Wireless, as well the few Alibaba IPv6 server addresses. For the rest of this chapter, except otherwise noted, we use the combined results of all techniques.

7.2.6 Limitations

The first limitation of our methodology relates to the stability of the IoT domain patterns. IoT backend providers constantly update their service infrastructure, so the patterns need to be regularly updated. Moreover, not all providers publicly release their documentation. When the documentation is not available; we do not try to identify those IPs due to ethical concerns.

The second limitation is that some providers might not use TLS for their services [262]. This might heavily impact the usefulness of TLS scans, such as the Censys dataset. This limitation motivated us to augment the scan data with DNS data. Still, even DNSDB has its own limitations, e.g., it does not have full coverage of all DNS requests. Third, we leverage passive traffic data from an ISP in Europe to analyze IoT traffic in the wild. Naturally, the vantage point's location might influence the overall IoT traffic that we see.

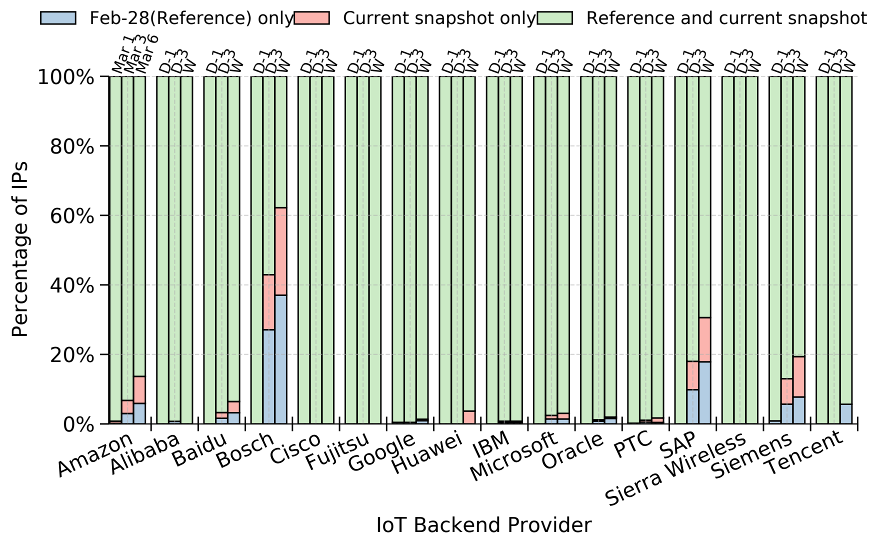


Figure 7.4: *IoT backend: Stability of server IP set.*

Finally, our ability to discover IPv6 addresses is directly influenced by the coverage of the chosen IPv6 hitlists.

7.3 IoT Backend Characterization

In this section, we provide insights regarding the deployment strategies of IoT backend providers for their Internet-facing gateways that enable the communication between the IoT devices and the backends' internal systems.

7.3.1 Stability of IoT Backends

Before we dive into the characterization of the IoT backend deployments, we evaluate how stable the set of discovered IoT backend server IPs—the gateways—is across time. This gives us information on how frequently we have to repeat our measurements. Using our weekly dataset, we, in Figure 7.4, we highlight changes in the daily IoT backend server addresses per IoT backend. Our reference date is the first day, Feb. 28, 2022. The first bar for each backend compares it to the next day, namely, March 1. We distinguish between IPs that are in both sets (green bar), that are newly discovered (red), and those that are only in the first set (blue). The other two bars are for March 3 and March 6.

We find hardly any change between the first two days. For most IoT backends, there is hardly any change within the week. This indicates that a weekly measurement suffices. However, there are some exceptions, i.e., Amazon AWS, Bosch, SAP, and Siemens. This is because these at least partially rely on shared public cloud infrastructure„ as we show later. Their IP set is more volatile, e.g., due to service scaling or service migration. However, this is not necessarily the case as some cloud

Backend Provider Name [Source]	# AS	# IPv4 /24 (IPv6 /56)	# Locations	# Countries	Protocols (Ports)	Strategy
Alibaba IoT [253, 257, 264]	2	73 (2)	27	13	MQTT(1883), HTTPS(443), CoAP(5682)	DI
Amazon IoT [47, 181, 265]	4	9,000 (20)	18	15 +Anycast	MQTT(8883, 443), HTTPS(443, 8443)	DI
Baidu IoT [254, 266, 267]	2	26 (1)	2	1	MQTT(1883, 1884, 443), HTTP(80, 443), CoAP(5682, 5683)	DI
Bosch IoT Hub[246]	1	290 (0)	1	1	MQTT(8883), HTTPS(443), AMQP(5671), CoAP(5684)	PR
Cisco Kinetic[268, 269]	2	14 (0)	4	2	MQTT(8883, 443), TCP(9123, 9124)	PR
Fujitsu IoT [247]	1	2 (0)	2	1	MQTT(8883), HTTPS(443)	DI
Google IoT core[242, 270]	1	114 (11)	77	14	MQTT(8883,443), HTTPS(443)	DI
Huawei IoT[249]	1	26 (0)	2	1	MQTT(8883, 443), HTTPS(8943), CoAP(NA)	DI
IBM IoT [250, 271]	2	116 (0)	12	8	MQTT(8883, 1883), HTTP(S)(80,443)	DI
Microsoft Azure IoT Hub[16, 272]	1	282 (0)	39	16	MQTT(8883), HTTPS(443), AQMP(5671),	DI
Oracle IoT[251, 273]	3	67 (0)	10	8	MQTT(8883), HTTPS(443)	DI+PR
PTC ThingWorx[274]	3	881 (0)	10	8	Protocol Agnostic	PR
SAP IoT[252, 275]	6	2.929 (0)	7	5	MQTT(8883), HTTPS(443)	PR
Siemens Mindsphere [248, 276]	4	126 (1)	3	3 +Anycast	MQTT(8883), HTTPS(443), OPC-UA	PR
Sierra Wireless [277–279]	4	7 (2)	4	4	MQTT(8883,1883), HTTP(S)(80,443), CoAP(5682,5686)	PR
Tencent IoT [258, 280]	5	47 (2)	5	4	MQTT(8883,1883), HTTP(S)(80,443), CoAP(5684)	DI

Table 7.2: *Selected IoT backends (alphabetical order) and their base characteristics for the study period, Feb. 28–Mar. 7, 2022. Dedicated Infrastructure (DI), Public Cloud Resources, or CDN (PR). We have released the set of domain patterns under the following link [281]*

providers [263] offer static IPs. As such, the IoT backend IP usage also depends on the IoT company strategy. We use all IPs discovered during the weekly study period for the remainder of this section.

7.3.2 Footprint

For the following reasons, it may be important for an IoT backend provider to have a presence in multiple physical locations: First, having a footprint in multiple data centers minimizes the impact of outages, physical disasters, or attacks on a subset of them. Second, datacenters from different regions are useful for coping with regional demands and can improve application performance. Third, it is increasingly important that data centers for IoT backends are available in different regions to comply with regulations regarding transferring, processing, and storing data. For example, in the EU, the General Data Protection Regulation (GDPR) poses constraints regarding data leaving EU borders.

We use a number of heuristics to infer the footprint of each IoT backend. Many of the IoT backends, e.g., Google [282] and Baidu [267], encode the location in the domain name. Typically they use city level, e.g., two or three letters, or airport codes. Others, e.g., Amazon [47], Alibaba [283], and Huawei [284], use region codes in the domain name that can be mapped to cities using their documentation. Using such hints, we are able to determine the footprint of all IoT backends, except Oracle and a small subset of IPs. For these, we use multiple sources, including the location of the prefix announcements from Hurricane Electric, Censys geolocation information, and pings from traceroute looking glasses to locate each IoT backend server IP. Typically, all alternatives point to the same location. In less than 7% of cases, these sources report different locations, in which case we use the majority vote.

The results, see Table 7.2, show that most IoT backend providers use multiple locations in at least two countries. However, there are exceptions: Baidu's and Huawei's backends are located only in China. This is surprising given that Baidu and Huawei operate data centers worldwide. Still, our extensive analysis allows no other conclusion.

Bosch offers a diverse range of IoT-related products, including machine learning, data analysis, and device management. These components rely on multiple public cloud providers in multiple locations around the globe. Each component has to be purchased individually. The Bosch IoT Hub component is the only one that offers a frontend for IoT devices. Therefore, we restrict our study to the locations and affiliated servers of the Bosch IoT hub. It has a single location.

IoT backend providers use different deployment strategies ranging from using *Dedicated Infrastructure (DI)* to *Public Cloud Resources (PR)*. We say that an IoT backend uses DI if all its identified IP addresses are announced by an autonomous system that is managed by the backend. If the IP addresses are announced by a cloud provider or CDN, we refer to it as PR. Of our 16 IoT backend providers, nine rely on dedicated infrastructure while six rely on public cloud providers, i.e., PTC relies on the AWS and Microsoft clouds, Siemens relies on AWS, Microsoft IoT services, as well as Alibaba. Such diversity enables providers to improve their footprint and offer services in many regions around the globe. The last IoT backend provider–Oracle–expands his own dedicated infrastructure by leasing resources from Akamai (we label this as DI+PR).

7.3.3 Network Diversity

First of all, we notice that the use of IPv6 is relatively low, and we can only discover IPv6 IoT backend server addresses for only seven of the 16 IoT backend providers. Hereby, Alibaba offers IPv6 only in China, and Microsoft explicitly states in its documentation that it does not yet support IPv6. Overall, the number of discovered addresses is significantly smaller for IPv6 than IPv4, see Table 7.2.

Network diversity, i.e., reachability of IoT backends via multiple ASes and/or prefix diversity, is important to circumvent congestion, blocking, and network misconfiguration to enable fast reroute and improve performance. We use the Routeviews Prefix to AS mapping dataset from CAIDA [285] to map IP addresses to prefixes and AS numbers. Our analysis shows that all IoT backend providers in our study use multiple, in some cases, tens of prefix advertisements, typically from more than one AS. Thus, we can expect that short-term routing or availability disruption leads to minor service degradations. Indeed, given the many available IPs and prefixes, it should be possible to use DNS to redirect IoT requests to available and well-performing IoT backend servers. We revisit this hypothesis when studying a large-scale outage of one of the largest cloud providers in Section 7.5.

Six IoT backend providers, namely Bosch, PTC, Siemens, SAP, Sierra Wireless, and Cisco, rely on one or more public cloud providers. This enables them to cope with the short-term unavailability of outages. Also, as mentioned earlier, Oracle uses its own dedicated infrastructure as well as that of a CDN. At least two of the IoT backend providers, Amazon IoT and Siemens, also use *anycast* or, more specifically, the Amazon Global Accelerator service [286]. Anycast services aim to map IoT requests to servers close to the client and cope with disruptions. This highlights that IoT backend providers care about reliability and diversity.

7.3.4 Protocol Support

In Table 7.2, we also report–per IoT backend provider–the supported protocols as listed in their documentation. They all claim to support MQTT, an often used protocol for IoT messaging. The protocol is lightweight, follows the publish-subscribe paradigm, and is designed for machine-to-machine communication. However, the IoT backend providers use different MQTT ports. Some use the default unencrypted MQTT port 1883. The majority uses the encrypted MQTT port 8883. Other providers also use non-standard ports, i.e., non-IANA assigned to a protocol. For example, for MQTT Baidu listens to port 1884. At least three IoT backends, i.e., Amazon, Baidu, and Google abused the secure Web port 443 for MQTT.

In addition, they often offer support for other IoT-specific protocols, including CoAP and AMQP. The ports vary, e.g., include 5682 and 5684 for CoAP. Baidu supports

CoAP requests on multiple ports, i.e., 5682 and 5683. AMQP is the least popular protocol among our IoT backend providers and is offered on port 5671. We also observe some application-specific protocols, e.g., Siemens offers OPC-UA, while PTC offers a protocol agnostic communication platform. The majority of the IoT backend providers also support Web protocols, namely HTTP on port 80 and/or HTTPS on port 443.

We conclude that non-expectedIoT backend providers quite often use non expected ports. Thus, purely probing the expected ports can be misleading. This is in line with recent results that observed unexpected applications running on servers [20]. The motivation for offering different ports even for the same IoT protocol, e.g., MQTT, may be to circumvent port blocking. This is likely the reason why MQTT service is offered at port 443 by some of the providers [270].

7.4 IoT Traffic Flows

So far, we have used our methodology to understand the footprint of the IoT backends. Next, we use traffic information from a large European Internet Service Provider (ISP) to study IoT traffic patterns.

7.4.1 Vantage Point

Our vantage point is a major European ISP offering residential Internet IPv4 and IPv6 connectivity to more than fifteen million broadband subscriber lines. The ISP uses NetFlow [50] to monitor the traffic flows at all border routers of its network, using a consistent sampling rate across all routers. This data is needed to support daily operations as well as network planning. For the ISP analysis, we anonymize all IoT company names.

Study Periods. For our IoT traffic flow analysis, we match the study periods for which we identify the footprint of the IoT platform providers, i.e., February 28 to March 7, 2022. In addition, we do a focused study during an outage, see Section 7.5, for December 3–10, 2021.

7.4.2 IoT Backend Platforms: Visibility

Our characterization of IoT backend providers has shown that they often rely on a global footprint to offer their services globally. Our first analysis, thus, focuses on the visibility of the IoT backend servers from our vantage point, i.e. the European residential ISP.

Our first validation check is whether any servers are within the address space of the residential ISP. This applies to none, which is expected as we study the traffic of subscriber lines. Our next check is for IoT backend infrastructure visibility from our

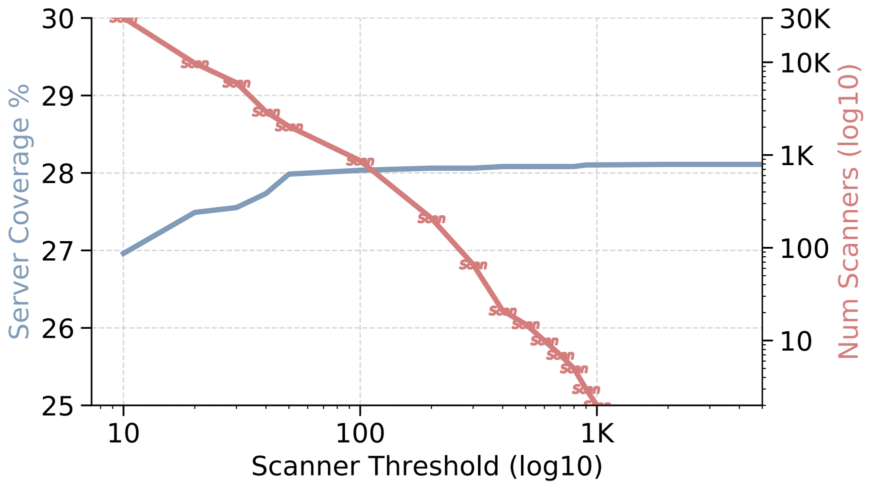


Figure 7.5: *Scanner threshold vs. % IPv4 IoT backends (blue line, left y-axis) and # scanning subscriber lines (red line, right y-axis (log)).*

vantage point, i.e., which fraction of the identified backend server IPs are contacted by subscriber lines of the ISP. Hereby, we do not expect that all servers are contacted as traffic localization, and other operational criteria within the IoT backends should map the ISP subscriber lines to a subset of their servers.

Exclusion of Scanners–Global Visibility. However, before proceeding, we have to exclude potential scanners within the ISP since their scan traffic may bias our estimation of the visible part of the IoT backend infrastructure. Scanners typically scan all or a substantial fraction of all IPv4 IPs, resp. IPv6 IPs of the IPv6 hitlist. Therefore, a subscriber line with a scanner is expected to send traffic to all IoT backend servers. Therefore, we exclude scanners from our analysis which is possible as the ISP uses spoofing prevention according to BCP38 [138].

To identify scanners, we follow the method proposed by Richter *et al.* [287]. For each day during our study period, we compute the fraction of IoT backend server IPs that a subscriber line is contacting. A subscriber line is said to host a scanner if it contacts more than a threshold of many of the server IPs. Figure 7.5 shows the results both for server coverage as well as ISP subscriber lines with scanners for February 28, 2022. More precisely, we show how this fraction changes as we increase the strictness of our criteria for identifying scanners—the scanner threshold (x-axis). Hereby, our minimum scanner threshold is 10 IoT backend server IPs—a very strict selection criteria. We see that as we increase the scanner threshold, the number of scanners (red line and right y-axis) decreases substantially. Yet, the percentage of IoT backend servers that are visible does not increase drastically (blue line and left y-axis).

We consider some baseline numbers: with a scanner threshold of 10, roughly 27% of all identified IoT backend servers are visible while removing about 30k subscription lines. Using a threshold of 100 leads to the removal of less than 800 subscriber lines per day while resulting in visibility of IoT backend servers of approximately 28%.

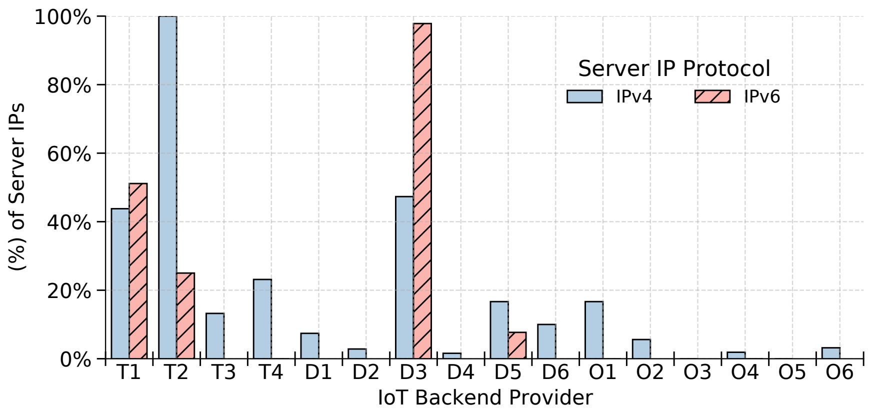


Figure 7.6: *ISP vantage point: % of Server IPs per IoT backend platform (Scanner threshold 100).*

As households often deploy multiple IoT devices contacting 10 backend IoT IPs is still reasonable, as underlined by the large number of subscriber lines. However, 100 server IPs are unlikely. As such, for the rest of the chapter, we use a scanner threshold of 100, which results in a daily visibility of roughly 28% of the identified IoT backend server IPs for IPv4 and 51% for IPv6 during our study period. Using this data, we identify more than 2.32 million IPv4 and 202K IPv6 ISP subscriber lines with IoT activity per day.

Visibility per IoT Backend Provider. Next, we investigate if the visibility of IoT backend server IPs is uniform across IoT platform providers. In Figure 7.6 we plot the percentage of visible servers for each platform for IPv4 as well as IPv6. As expected, the visibility varies substantially across the IoT backend providers. For most, it is relatively small, between 5% to 20%. As remote IoT backend servers should not be contacted by subscription lines from a European residential ISP, this is to be expected. Recall our insights from Section 7.4.7 about the locations of the discovered IoT server IPs. Surprisingly, for two IoT backend providers, namely T1 and D3, we observe around half of the discovered IoT backend server IPs. Moreover, for one IoT backend provider, namely T2, almost all IoT backend server IPs are visible. This provider is also among the top 4 popular providers. On the other hand, for two other platform providers, namely O5 and O3, we hardly find any activity. Since they are not focusing on the European residential market, we exclude them from our analysis in this section.

7.4.3 ISP Subscriber Line Activity by IoT Backend Platform

We find that a substantial fraction of ISP subscriber lines contact IoT backend platforms. This underlines that the residential ISP is a suitable vantage point.

ISP Subscriber Lines–Visibility by Data Source. While we know that our different data sources increase the discovery of IoT platform server IPs, we do not yet know how important this is for discovering IoT traffic. Thus, we check the necessity

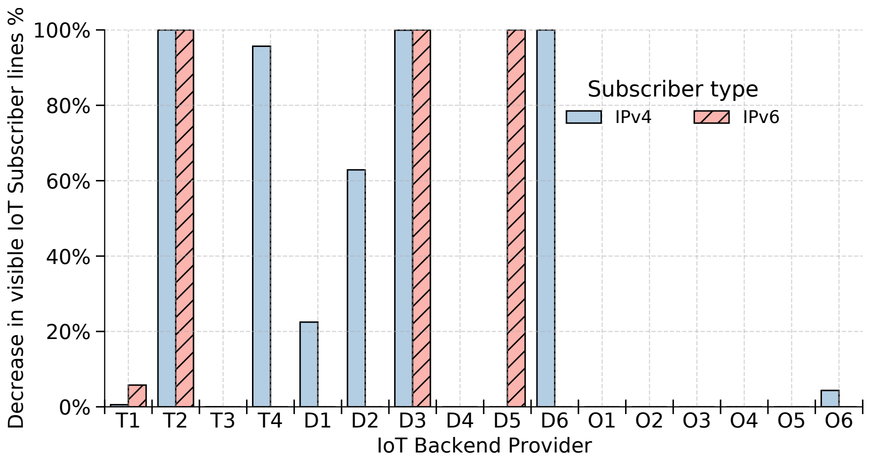


Figure 7.7: *ISP vantage point–per IoT platform: % decrease in ISP IoT subscriber lines by considering only TLS certificates.*

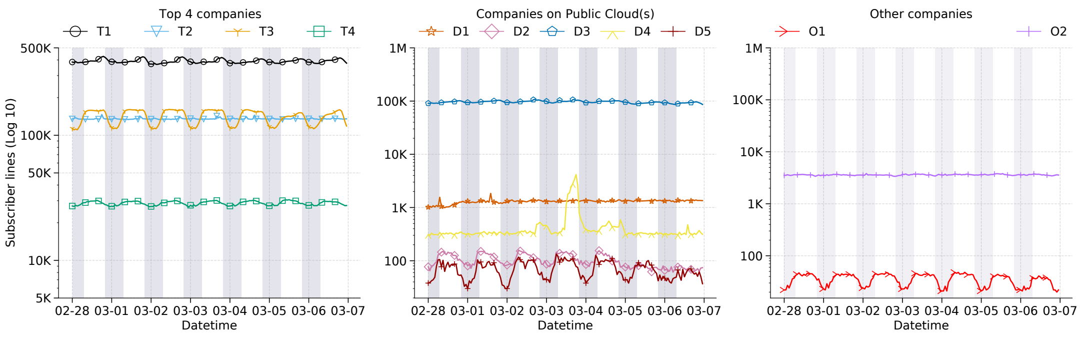


Figure 7.8: *ISP vantage point–per IoT platform: # of active subscriber lines.*

of using different data sources, namely TLS certificates vs. passive and active DNS data. For this, we plot, in Figure 7.7, the decrease in discovered subscriber lines with IoT traffic when we rely only on TLS certificate information gathered by active IP scans (the Censys data set). For some IoT platform providers, e.g., T4, D6, T2, and D3, almost none of the subscription lines would have been detectable. Note that two of these are providers that rely on SNI.

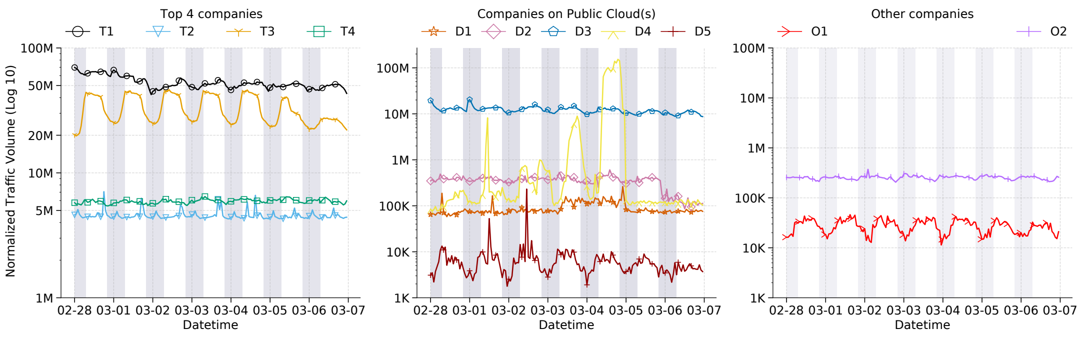


Figure 7.9: *ISP vantage point–per IoT platform: Normalized total downstream traffic volume.*

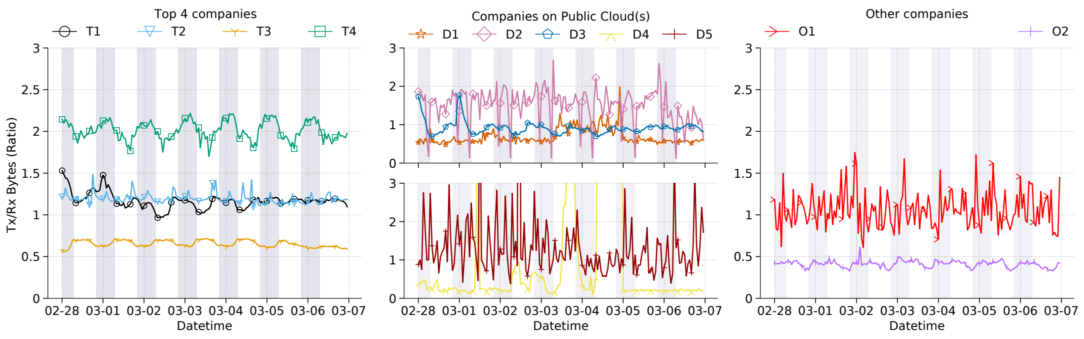


Figure 7.10: *ISP vantage point–per IoT platform: Ratio of Downstream to Upstream traffic.*

ISP Subscriber Lines–Activity across Time. Next, we explore how ISP subscriber line activity changes during our study, see Figure 7.8. It plots the hourly number of subscriber lines for each IoT backend provider across the week. To plot the subscriber line activity, we consider three subgroups of IoT backend providers, namely, the top-4 per revenue, the ones that depend on cloud providers, and the remaining ones. We only include those with at least 15 subscriber lines per hour.

Figure 7.8 (left) shows the activity of the top-4 IoT backend providers. We use a light shading for the night–8 pm to 8 am local time–to help in identifying the time of day effects. First, the level of subscriber line activity differs significantly—in fact, by orders of magnitude. Some have a clear diurnal pattern, e.g., T3, while others, e.g., T2, are more or less constant. We also observe that the peak time differs among these IoT backend providers. The peak time for T1 and T4 is during prime time, i.e., between 6-10 pm, while for T3, it is constant during the day, i.e., between 8 am and 8 pm. We attribute this to the type of services that IoT devices offer and how often they communicate with their IoT backend providers. For example, some IoT devices are likely to be used at home for entertainment during prime time, while others offer services that are used at any point in time.

Next, we move to those IoT backend providers that rely on the public clouds, see Figure 7.8 (center). Again, we see a large difference in their usage across the board. Moreover, their activity does not correlate to one of the platform providers (plot not shown). Similar observations hold for the remaining IoT backend providers, see Figure 7.8 (right).

7.4.4 IoT Backend Traffic

Next, we look at traffic levels. Here, we observe similar patterns as in the IoT subscriber lines analysis, which is expected as many of the IoT applications are triggered by subscriber lines activity.

IoT Backend Traffic–Downstream Volume. We find that the relative traffic volume level changes substantially, see Figure 7.9. It shows the normalized downstream traffic volume for the same groups of IoT backend providers as before, namely, top 4,

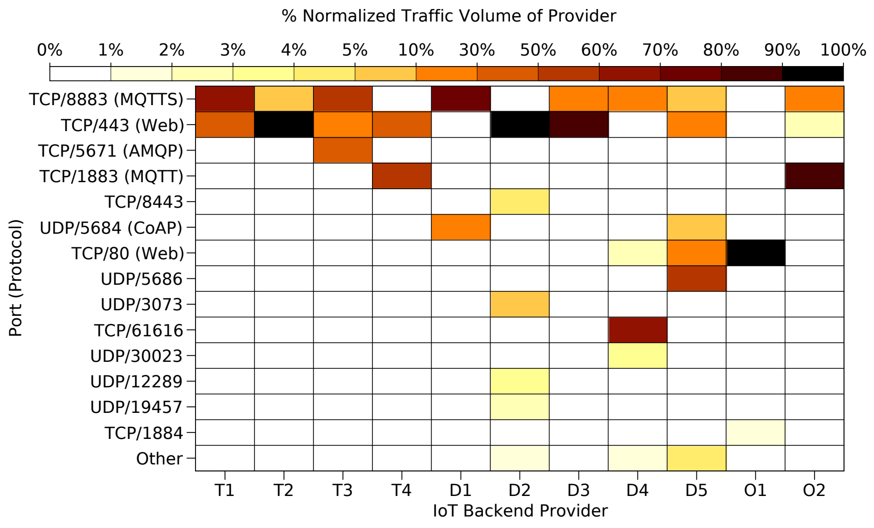


Figure 7.11: *ISP vantage point–per IoT platform: % traffic volume per port.*

public-cloud-dependent, and others. We notice that the traffic volume per subscriber line differs substantially. On the one hand, even though the number of observed subscriber lines differs by an order of magnitude for T1 and T3, their total traffic levels are relatively close. On the other hand, even though T2 and T3 serve a similar number of subscriber lines, their traffic volume differs by more than a magnitude. This is because the traffic demands of IoT devices depend on the applications. Thus, we conclude that the number of subscriber lines served by IoT backend providers is not a good indicator for the downstream traffic volume level of the provider. This holds for all IoT backend providers that we study.

IoT Backend Provider Traffic–Traffic Ratio. We also notice that IoT applications' downstream and upstream traffic demands differ. Some IoT applications are heavy-upstream, e.g., video surveillance, while others are heavy-downstream, e.g., online media streaming. This is reflected in the IoT backend traffic. In Figure 7.10, we plot the ratio of downstream vs. upstream traffic for the IoT backend providers of our study. Values above 1 indicate that the IoT backend provider sends more traffic to the IoT devices than it receives. Our analysis highlights that IoT backend providers differ. In all three groups, namely, top-4, public cloud dependent, and the rest, we can find heavy downstream as well as heavy upstream ones. Indeed, there is no particular pattern to it. The ratios range from less than 0.33 to more than 3, which shows that the asymmetry in the downstream vs. upstream ratio is significant. Moreover, we do not notice correlations between the ratios, the number of observed subscriber lines, or the downstream traffic level.

7.4.5 IoT backend provider–Port Usage

Next, we explore which network ports the IoT devices are using. Are they relying on general-purpose application layer network protocols such as HTTP or HTTPS, or are they using IoT-specific application protocols? Accordingly, Figure 7.11 shows the application layer protocol mix as identified by IANA assigned port numbers for each IoT backend provider, i.e., the percentage of traffic for each application protocol. Again, there is no single pattern that describes all IoT backend providers.

Many utilize the popular Web secure ports, e.g., 443, typically over TCP. Its usage varies from 5% up to 90%. IoT-specific protocols, e.g., MQTT, are also popular. However, which MQTT port is used differs across IoT backend providers. IANA assigns port 1883 for the non-secure version and port 8883 for MQTT over TLS. However, recall, some IoT backend providers, as per their documentations, also offer MQTT service over non-standard ports like 1884 or even 443. The reasons for serving MQTT over non-standard ports include reduction of attack surface by reducing discovery probability via scans and circumvention of firewalls that block standard MQTT.

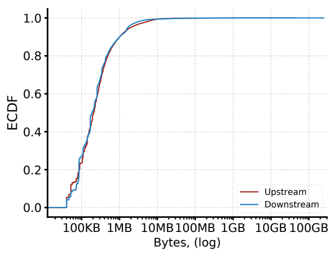
We find that secure MQTT over its standard port is quite popular and used by more than 50% of all studied IoT backend providers. Other popular IoT protocols include CoAP and AMQP. Similar to MQTT, some providers offer CoAP over non-standard ports, e.g., the neighboring ports 5686 and 5682. For UDP/5686, we do observe the activity. For one provider, namely D4, we see that it exchanges substantial traffic volume over port TCP/61616. This port number is the default port number of the popular messaging software, Apache ActiveMQ [288], which processes messages sent via IoT-specific protocols such as MQTT and AMQP. We further observe a number of UDP ports above 10000 in use by various IoT backend providers.

Overall, this diverse port usage confirms previous insights [20] that port scanning and protocol handshake do not suffice to uncover the IoT backend server infrastructure. In addition, to capture IoT-related protocols, it is not sufficient to aggregate traffic of IoT-specific protocols, as this misses a substantial part of the IoT traffic, e.g., the one served using HTTP(s) ports.

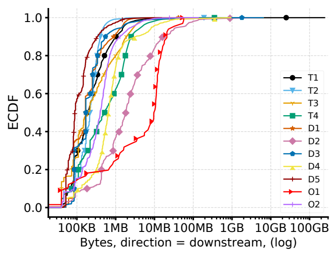
7.4.6 Traffic Characteristics

We also investigate the characteristics of the traffic exchanged between subscriber lines and IoT backend providers. This is an important test to validate that this traffic is not generic Web traffic or video streaming of popular applications that can be misinferred as IoT-related traffic.

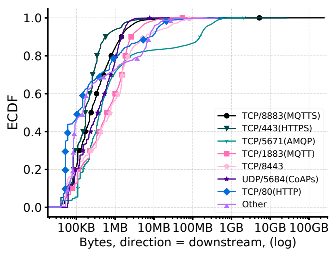
In Figure 7.12a, we plot the empirical cumulative distribution function (ECDF) of the estimated traffic exchanged in a day between a subscriber line and all the IoT backend providers we consider in our study. We estimate the exchanged traffic considering the sampling rate. We plot both the download and upload traffic exchanged, as some applications may be download-dominant or upload-dominant. Our analysis shows that for the vast majority (more than 99%) of the subscriber lines, both the upload and download traffic exchanged with all the IoT providers is less than 10 MB per day. This value is substantially lower than the reported traffic consumed by smart TVs or residential users, which is no less than 1 GB per day [28, 289, 290]. Thus,



(a) *Estimated traffic exchanged in a day between a subscriber line and all IoT backends in our study.*



(b) *Estimated traffic exchanged in a day between a subscriber line and each IoT backend in our study. In this plot, we consider the download traffic per subscriber line.*



(c) *Estimated traffic exchanged in a day between a subscriber line and all IoT backend providers for popular ports. In this plot, we consider the download traffic per subscriber line.*

Figure 7.12: *ISP vantage point: Traffic characteristics for traffic exchanged in a day between a subscriber line and IoT backend providers or popular ports in our study.*

we conclude that the traffic exchanged between subscriber lines and IoT providers is unlikely to be general Web or popular application video traffic.

Then, we investigate if any of the IoT providers we consider in our study deviates from the above mentioned behavior and offers general Web or popular video streaming applications in the identified prefixes. In Figure 7.12b, we plot the empirical CDF for the estimated traffic exchanged in a day between a subscriber line and each of the IoT backends we consider in our study for the download traffic volume. Although there are differences across IoT providers, the general observation is that the vast majority of the exchanged traffic is relatively low, i.e., less than 10 MB per day. Thus, the IoT backend servers for each of the IoT backend providers we consider in our study are unlikely to be used for general Web or popular video traffic. Similar observations are made when we analyze the upstream traffic.

Finally, we investigate if the traffic exchanged using specific ports indicates the exchange of heavy traffic. In Figure 7.12c we plot the traffic exchanged between subscriber lines and IoT backend providers for the most popular ports in our study. We consider the downstream direction and the top-7 ports that contribute to more than 95% of the exchanged traffic and the aggregation of the rest of the ports. Our analysis shows that there is only one port, namely, port 5671 (this port is registered with IANA for the secure version of the AMQP protocol), where around 18% of the subscriber lines exchange between 100 MB and 1 GB per day. The high traffic volume exchanged is observed only in one of the IoT providers, and it is a very small fraction of the overall traffic we observe in our measurements. Similar observations are made when we analyze the upstream traffic. We conclude that the vast majority of the traffic exchanged at different ports between the subscriber lines and IoT backend providers do not resemble the general Web or popular video traffic.

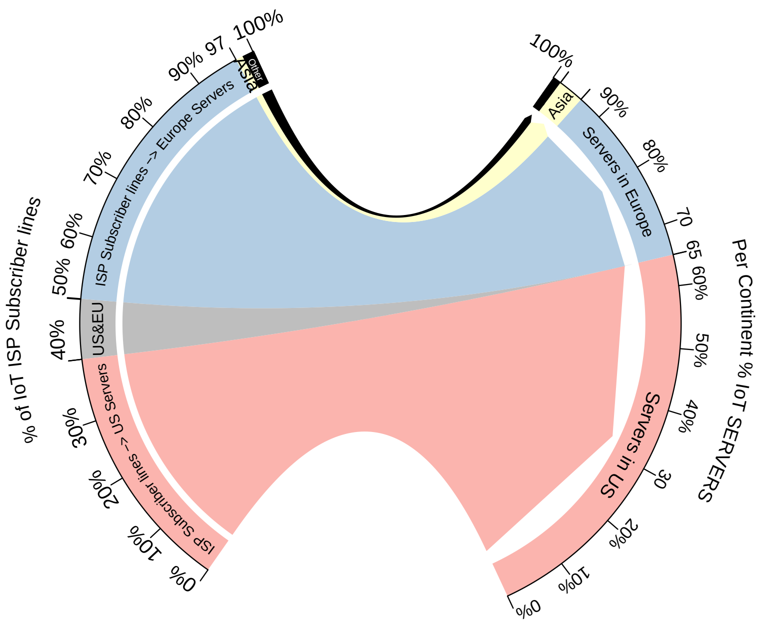


Figure 7.13: *% of ISP Clients communicating with % of Servers in each continent.*

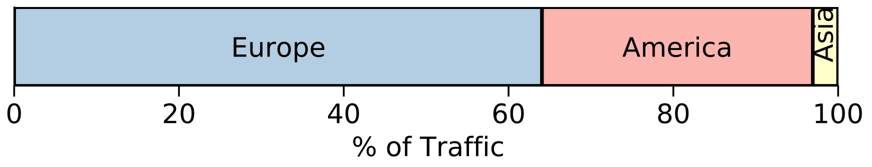


Figure 7.14: *% of ISP Traffic communicating with % of Servers in each continent.*

7.4.7 Crossing Region Borders

Since the recent EU General Data Protection Regulation (GDPR) poses restrictions on the transfer of data outside the EU, and since the transfer of data to remote servers may impact the performance of delay-sensitive applications we next study how many of the European ISP's subscriber lines with IoTs contact IoT servers outside of Europe. Hereby, we take advantage of the location information collected for each IoT backend server IP.

In Figure 7.13 we visualize the percentage of IoT-hosting subscriber lines, see left-side of plot, that exchange traffic with IoT backend servers in different regions, namely, Europe, the US, Asia, and others. Our analysis shows that slightly less than half, i.e., around 47% of the IoT-hosting subscriber lines communicate exclusively with IoT backend servers located in Europe. Around 40% of the IoT-hosting subscriber lines contact IoT backend servers in the US. Around 10% of the IoT-hosting subscriber lines contact a mix of locations from the EU and US. Around 3% of the IoT-hosting subscriber lines are contacting only IoT backend servers in Asia or other regions.

On the right-hand side of the plot, we visualize the percentage of IoT servers hosted per continent. We see that the IoT backend servers in Europe are a minority of

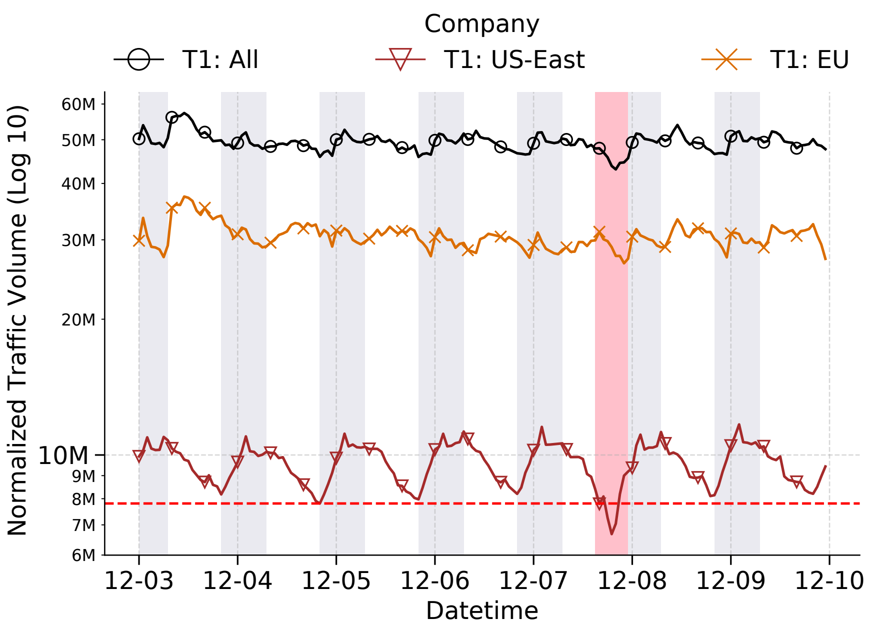


Figure 7.15: *ISP vantage point–IoT backend provider T1: Normalized downstream traffic volume for all US-East, and EU service regions. The AWS outage is highlighted using a red background. The red line shows the normalized minimum traffic volume for the US-East of the previous week.*

contacted servers, only around 30%. Indeed, the majority of the IoT backend servers, i.e., around 65%, are located in the US. Around 5% of the IoT backend servers are located in Asia and a tiny fraction elsewhere. We conclude that around half of the IoT-hosting subscriber lines in the European ISP contact IoT backend servers located in Europe, although they account for less than one-third of the IoT backend servers identified in our study.

Regarding exchanged traffic volume between subscription lines and IoT backend servers, we notice that most of the traffic stays in Europe. In Figure 7.14, we plot the percentage of traffic exchanged between subscription lines and IoT backend servers annotated by location. The largest traffic fraction, more than 62%, is exchanged between subscriber lines in Europe with servers in Europe. However, around 35%—a substantial fraction—is exchanged with servers in the US (where the majority of the IoT backend servers are located). As such, IoT traffic is less localized than one may have expected, given the regulations of GDPR.

7.5 IoT Backend Disruptions

Here, we consider actual as well as potential disruptions.

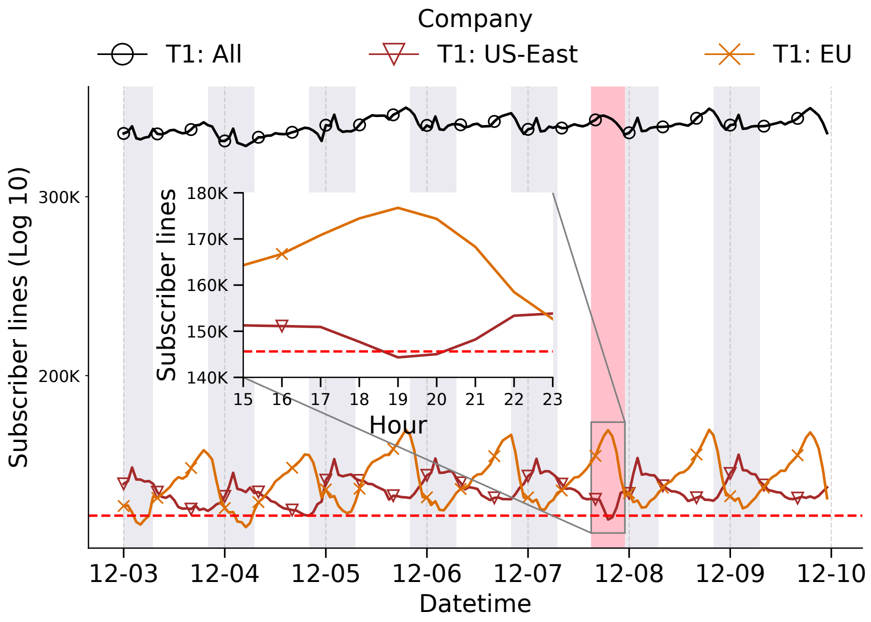


Figure 7.16: *ISP vantage point–IoT backend provider T1: # of subscriber lines for all, US-East, and EU service regions. The AWS outage is highlighted using a red background. The red line shows the minimum # of subscriber lines for US-East of the previous week.*

7.5.1 AWS Outage

During the time when we collected preliminary results (Dec. 3–10, 2021), a major outage happened within the infrastructure of one of the major cloud providers. More precisely, on December 7, 2021, Amazon Web Services, a cloud provider that is heavily used by the IoT backend providers that we study, experienced a large-scale outage [291–294] of its US-East-1 service region (located in Northern Virginia). This outage affected many popular websites and Internet services. Thus, we examine the effect of this outage on the traffic flows of the IoT backend providers.

Impact on T1 ISP traffic flows. First, we analyze the outage's effect on the traffic from the T1 IoT platform to the ISP's subscribers. Figure 7.15 shows T1's normalized downstream traffic volume towards the ISP as well as the normalized volume for two different AWS service regions, namely US east and EU–aggregating the traffic of all US east resp. EU availability zones. During the outage (highlighted via the red background) there is a substantial traffic drop–more than 14.5% for the US east coast region. Indeed, the total traffic volume is substantially lower than the minimum observed traffic volume of the previous week (red line). This highlights that cloud outages such as the one by AWS do not only affect Web services [292–294] but also IoT services [295, 296]. When looking at the total traffic as well as the traffic from the EU sites, we notice only slight dips. One reason is that the EU region services more than three times as much traffic as the US east coast region.

Still, the fact that there is a drop even in traffic in the EU region indicates some interdependencies between the regions.

Impact on T1 subscriber lines. Next, we check how these traffic volumes relate to the number of ISP subscriber lines that contact IoT servers in these AWS regions, see Figure 7.16. Again the red background highlights the outage, and the horizontal line corresponds to the minimum number of subscriber lines of the previous week. We see no impact for the EU region but a slight decrease for the US East coast region. One may ask why this decrease is so small. The reason is that we still observe the attempts of the IoT devices to contact the servers in their assigned AWS regions. Thus, the downstream traffic is lower, but the number of subscriber lines does not change drastically. Still, it decreases, which indicates that some of them stopped trying, we did not observe them due to their decreased traffic volume, or they are remapped to other regions.

Impact on D1–D6. Next, we explore whether the outage affected the IoT backend providers relying on AWS or the T1 IoT platform. We find hardly any effect as the subscriber lines of these platforms are mainly mapped to the EU AWS regions.

7.5.2 Potential Disruptions

Possible disruptions that we study for the week starting in Feb. 2022 are connectivity problems due to routing and or IP filtering based on blocklists.

Connectivity problems. Such problems include routing problems such as BGP leaks, BGP hijacks, and AS outages. We rely on Cisco's BGPStream service, which provides historical information about BGP hijacks, leaks, and outages [297]. It identified 10 BGP leaks, 40 possible BGP hijacks, and 166 AS outages. None of these affected any identified IoT backend server IPs or the ASes they are hosted in.

IP Filtering. Next, we check how likely it is that a backend becomes unreachable as a consequence of appearing in a blocklist. Here, we take advantage of the FireHOL project[298] which generates a list of suspicious addresses, by combining information from popular blocklists. In Feb. 2022, the FireHOL blocklist contained over 610M IPv4 addresses extracted from 67 blocklists[2]. Using daily blocklists matching our study period, we check if the server IPs are included in any of the blocklist. We identified 16 such IPs. The non-exclusive reason for their inclusion in the blocklist are: four are associated with open-proxies and anonymizing services, one is linked to malware, and five are associated with network attacks/spam. Moreover, nine originate from a personal blocklist[3]. These IPs belong to 6 of our IoT backend providers, namely, Baidu (5 IPs), Microsoft (4 IPs), SAP (4 IPs), Google (3 IPs), Amazon (2 IPs), and Alibaba (1 IP).

[2]We excluded one of the blocklist as it is known that it is not carefully maintained, see https://github.com/pushinginertia/ip-blacklist/issues/9, and is, thus, likely to produce false positives.

[3]https://graphiclineweb.wordpress.com/tech-notes/ip-blacklist/

7.6 Chapter Summary

In this chapter, we developed and applied a methodology to characterize the deployment and traffic patterns of a substantial part of the IoT backend server infrastructure: IoT backend providers. Our techniques relied on a fusion of information from public documentation, passive DNS, and active measurements. Some sources are publicly available, and others are collected through active or passive measurements. The IoT backend provider market is consolidated, with 90% of the revenue going to the top 20 companies [299]. Our study focused on 16 IoT backend providers, including the top 10.

We showed that detecting the IoT backend provider's Internet-facing infrastructure is challenging; a pure IP-port scanning on the Internet misses a significant share of the IoT backend providers' addresses. Moreover, multiple backend providers use non-standard ports to serve IoT-related protocols.

Our study unveiled a multitude of deployment strategies among backend providers. Most have footprints in many geographical regions, and some utilize anycast. However, some offered their services in one location, and some only offered parts of their services to other regions. Our insights aid prospective customers in selecting appropriate IoT backend providers, as they can compare different deployment strategies and their performance implications.

We also studied the traffic patterns of the backend providers in our ISP vantage point. Our study showed that more than 40% of the IoT subscribers exchanged traffic with servers on different continents. Regarding traffic volume, around 30% of IoT traffic goes to servers outside of Europe, which raises questions about regulatory compliance, reliability, and performance. Moreover, we found that both the IoT population and activity per application differ vastly. Some IoT applications' traffic peak during the day while others peak in the evening hours.

Our study showed the dependency among the providers: with six of them relying on another. Some providers have a footprint in locations where others don't. Thus, although competing, some providers rely on others to expand their footprints. Such arrangements may have some reliability consequences. An outage in a provider may have a knock-on effect on the services of the relying providers. We conducted a case study on how an outage in a large public cloud provider can affect the services of other IoT backend providers. In our case, the outage happened in a region that did not affect our providers except one.

In Chapter 5, we developed signatures by observing traffic from IoT devices in a testbed. In this chapter, our lightweight methodology can complement the previous ones for purposes such as estimating the popularity of IoTs and identifying the IoT application activity in passive data. Our methodology is lightweight as it does not need to derive per IoT device/manufacturer signatures using, e.g., instrumented testbeds.

8 Conclusion and Outlook

IoT devices are becoming increasingly popular and provide a wide range of services to home and industry users. The IoT server backend infrastructures on the Internet are essential to many of the functionalities of IoT devices, and these devices and their server backend infrastructure collectively form the IoT ecosystem.

The IoT ecosystem is rapidly growing and evolving with an ever-increasing number of IoT devices and services. At the same time, it has been a victim and a source of significant security and privacy incidents. Characterizing this ecosystem at scale allows researchers, network operators, and other stakeholders to understand it better, keep track of its evolution, and develop solutions to handle its threats.

8.1 Summary

In this thesis, our overarching goal was to develop and apply the methodologies for characterizing the IoT ecosystem at scale. To achieve this goal, we considered (i) methodologies for detecting IoT devices in the wild using sparsely sampled flow data, (ii) the effect of the deployment of home IoT devices on users' privacy, and (iii) detecting and characterizing the IoT backend server infrastructure.

We collaborated with a large European Tier-1 ISP and a major European IXP to conduct our studies. For operational purposes, these providers routinely collect *large volumes* of network flow capture data such as NetFlow and IPFIX.

The large volume of network flow captures from ISP and IXP motivated us to study how to improve query response time while exploring large volumes of network flow captures. Thus, in Chapter 4 of this dissertation, we investigated this problem. We highlighted that network flow captures such as NetFlow and IPFIX are widely available, and they are essential for operators to monitor the health of their networks and steer their evolution. However, their analysis is time-intensive and challenging due to their ever-increasing size and complexity. In the past, this has substantially hindered ad-hoc queries across multiple sites, for different time periods, and over many network features. We designed, developed, and evaluated Flowyager, a system that allows exploration of network-wide data and answering ad-hoc apriori unknown queries interactively. It achieved this using existing network flow captures, without the need for specialized hardware and without compiling specific queries into telemetry programs that should be known in advance and are slow to update.

Next, in Chapter 5, we investigated how to detect home IoT devices without deep packet inspection or active measurements, both intrusive and unscalable methods for large deployments. We developed and evaluated a scalable methodology to detect IoT devices at subscriber lines with limited, sparsely sampled flow data from a major Tier-1 ISP and a large IXP, even if devices are not actively used. By classifying domains, and IP addresses of the backend infrastructure, we derived distinct signatures for recognizing IoT devices. With our signatures, we were able to recognize the presence of devices from 31 out of 40 manufacturers in our testbed. Our evaluation of methodology showed that 20% of 15 million subscriber lines used at least one of the 56. We highlighted that our technique scales and can identify millions of IoT devices within minutes. To this end, we discussed the potential security benefits of detecting IoT devices, why some IoT devices are faster to detect, how to hide an IoT service, as well as how the detectability can be used to improve IoT services and network troubleshooting. For security benefits, IoT detection can help the ISP/IXP identify what devices are common among the subscriber lines with suspicious traffic. Once identified, they can notify the device's owner or take further mitigating steps.

Then, In Chapter 6, we conducted a case study on how millions of deployed devices at homes can potentially endanger users' privacy. We showed a new way to defeat IPv6 privacy even when the ISP does prefix rotation, and all but one device at home use privacy extensions. We found that a single device that uses EUI-64 can be leveraged as a tracking identifier for devices with the same end-user prefix. This allows for longitudinal tracking, i.e., tracking across prefix rotations over multiple days. Our analysis showed that up to 19% of end-user prefixes in a large ISP could face IPv6 privacy leakage, and up to 17% of them could be monitored by third parties, primarily hypergiants. Our investigation unveiled that IoT devices and popular manufacturers contributed the most to this IPv6 privacy leakage. We proposed that vendors enable privacy extensions by default and that regulatory intervention is necessary to protect users' privacy.

In the last part of this dissertation, Chapter 7, we studied the IoT backend provider infrastructure. As the IoT device population and application complexity increase, a collection of IoT backend providers has been established to cope with the IoT application demands. These IoT backend providers are either IoT vendors or large cloud providers offering services tailored to IoT developer needs. Our study focused on 16 IoT backend providers, including the top 10. We developed a methodology to detect the Internet-facing section of their infrastructure. We argued that discovery of the Internet-facing part of the IoT backends is a challenging task as pure IP-port scanning misses a significant share of the addresses for many IoT backend providers. Indeed, we found that the port usage differs substantially across IoT providers. It is not unusual for IoT protocols, e.g., MQTT, to use non-standard ports or to reuse Web ports. The latter makes the identification of IoT backend infrastructure as well as IoT traffic challenging. However, fusing data from publicly available documentation, certificate data from active scanning, with passive and active DNS data allowed us to unveil a detailed map of IoT backend servers. Our study showed that IoT backend providers' deployment strategies differ substantially. While the footprints of most of them covered many geographical regions, some were present in only one location. Yet, others were utilizing anycast. Since this impacts service performance, it should also impact IoT backend provider selection. Moreover, regulatory compliance related to IoT data transfer, storage location, and processing also plays an increasingly important role when selecting an appropriate IoT backend. We analyzed the traffic patterns of the IoT backend providers at a major European ISP. Surprisingly, we found that around a third of the IoT traffic in our study was exchanged with servers in different continents, although it could have been served from within the region of the IoTs. This raises questions regarding the configuration of applications and best practices when developing IoT applications. It also raises questions regarding reliability. We found that a significant outage of a cloud provider impacted some IoT services. We also observed that six of the providers relied on another IoT backend provider to expand their footprint or outsource IoT backend functionalities. Thus, outages that occur unexpectedly can have cascading effects. For the one outage that we studied in detail, this did not happen as these providers used the regional service, which was not affected by the outage. Still, it is a wake-up call to add flexibility and re-routing opportunities to handle IoT backend disruptions, e.g., outages, attacks, misconfigurations, and blocklists. Our methodology also offers a scalable and lightweight approach for estimating the popularity of IoTs and sheds light on IoT application activity. This is possible without the need to derive per IoT device/manufacturer signatures using, e.g., instrumented testbeds. Our traffic analysis highlighted that the IoT population and activity per application differed vastly. While some applications behaved more like the typical user-generated traffic, i.e., diurnal patterns, peak evening hours, and were downstream-heavy, this was not the case for all IoT applications. In fact, some popular IoT applications' traffic peaked during the day.

8.2 Reflections

Internet Service Providers and large network operators have a responsibility to ensure the security of their networks. They are in a privileged position that enables to detect and mitigate large scale attacks that affect their subscribers or originate from them. Although, it is in their interest to secure and keep track of their ecosystem, it is also important that they use their position responsibly and avoid indiscriminate application of our methodologies and techniques on subscriber lines without tangible benefits to them. As such, it is crucial to apply such methods transparently while considering the interests of the subscriber lines.

Applying the methodologies and findings of these security measures has implications for subscribers, particularly from a privacy perspective. As a result, it is essential to consider the interests of subscriber lines when implementing these measures. A possible solution for both the provider and the subscriber is to bundle these security measures into IoT-security services. Providers can offer these services on an opt-in basis, either for free or even commercially. However, the implementation of such approaches would require extensive studies, and the details and business cases are beyond the scope of this thesis.

8.3 Future Work

The IoT ecosystem is growing and evolving; new security and privacy threats are emerging, which may endanger the health and operation of the IoT ecosystem and the Internet. Thus, we should keep monitoring its state and development and continuously challenge our assumptions. In future work, we intend to further scale our studies by cooperating with more vantage points and investigating the implications of the IPv6 transition on the IoT ecosystem.

More vantage points: We plan to collaborate with more ISPs and extend the number of our vantage points. IoT users worldwide may not use the same brand of devices; IoT companies may offer a region-specific version of their services to the users. In addition, the IoT penetration rate varies from region to region; hence, having more vantage points allows us to study and compare the characteristics of the IoT ecosystems in different regions.

IoTs in dual-stack networks: ISPs worldwide, though at different paces, are transitioning to IPv6, and some are already offering Internet connectivity in a dual-stack manner. We plan to investigate the behavior of IoT devices in dual-stack networks, e.g., measuring to what extent IoT devices and their backends support IPv6 and the security and performance implications of using both IPv4 and IPv6 on the IoT ecosystem.

IoT endpoint discovery in IPv6 Internet: IPv6's address space is vast; thus, Bruteforce active measurements that scan the whole IPv6 address space are infeasible. It is becoming more challenging to find IoT endpoints (devices or servers) on the IPv6 Internet through active measurements. At the same time, accessing passive data from large vantage points is not always possible for researchers. Thus, we need novel active measurement techniques to identify and track the transition of the IoT ecosystem to the IPv6 internet.

Bibliography

[1] Said Jawad Saidi, Anna Maria Mandalari, Roman Kolcun, Hamed Haddadi, Daniel J Dubois, David Choffnes, Georgios Smaragdakis, and Anja Feldmann. “A haystack full of needles: Scalable detection of iot devices in the wild”. In: *Proceedings of the ACM Internet Measurement Conference*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 87–100. ISBN: 9781450381383. DOI: 10.1145/3419394.3423650. URL: https://doi.org/10.1145/3419394.3423650 (cit. on pp. ix, 3, 6).

[2] Said Jawad Saidi, Srdjan Matic, Oliver Gasser, Georgios Smaragdakis, and Anja Feldmann. “Deep Dive into the IoT Backend Ecosystem”. In: *Proceedings of the 22nd ACM Internet Measurement Conference*. IMC ’22. Nice, France: Association for Computing Machinery, 2022, pp. 488–503. ISBN: 9781450392594. DOI: 10.1145/3517745.3561431. URL: https://doi.org/10.1145/3517745.3561431 (cit. on pp. ix, 7).

[3] 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. DOI: 10.1109/TNSM.2020.3034278 (cit. on pp. ix, 7).

[4] Said Jawad Saidi, Oliver Gasser, and Georgios Smaragdakis. “One Bad Apple Can Spoil Your IPv6 Privacy”. In: *ACM Special Interest Group on Data Communications(SIGCOMM) Computer Communication Review* 52.2 (June 2022), pp. 10–19. ISSN: 0146-4833. DOI: 10.1145/3544912.3544915. URL: https://doi.org/10.1145/3544912.3544915 (cit. on pp. ix, 7).

[5] Said Jawad Saidi, Damien Foucard, Georgios Smaragdakis, and Anja Feldmann. “Flowtree: Enabling Distributed Flow Summarization at Scale”. In: *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*. SIGCOMM ’18. Budapest, Hungary: Association for Computing Machinery, 2018, pp. 30–32. ISBN: 9781450359153. DOI: 10.1145/3234200.3234225. URL: https://doi.org/10.1145/3234200.3234225 (cit. on p. ix).

[6] Apoorv Shukla, Said Jawad Saidi, Stefan Schmid, Marco Canini, Thomas Zinner, and Anja Feldmann. “Towards Consistent SDNs: A Case for Network State Fuzzing”. In: *IEEE Transactions on Network and Service Management* 17.2 (2019), pp. 668–681. DOI: 10.1109/TNSM.2019.2955790 (cit. on p. x).

[7] IoT Analytics. *IoT 2019 in Review: The 10 Most Relevant IoT Developments of the Year*. https://iot-analytics.com/iot-2019-in-review/. 2020 (cit. on p. 1).

[8] Alex Schiffer. *How a fish tank helped hack a casino*. https://www.washingtonpost.com/news/innovations/wp/2017/07/21/how-a-fish-tank-helped-hack-a-casino/. 2017 (cit. on p. 1).

[9] M. Antonakakis et al. “Understanding the Mirai Botnet”. In: *USENIX Security Symposium*. 2017 (cit. on pp. 1, 27).

[10] Dan Goodin. *Rash of in-the-wild attacks permanently destroys poorly secured IoT devices*. `https://arstechnica.com/information-technology/2017/04/rash-of-in-the-wild-attacks-permanently-destroys-poorly-secured-iot-devices/`. 2017 (cit. on p. 1).

[11] Y. Jia et al. “Burglars’ IoT Paradise: Understanding and Mitigating Security Risks of General Messaging Protocols on IoT Clouds”. In: *S&P*. 2020, pp. 465–481 (cit. on pp. 1, 3, 27, 100–102).

[12] *Gartner Says Worldwide IoT Security Spending Will Reach $1.5 Billion in 2018*. `https://www.gartner.com/en/newsroom/press-releases/2018-03-21-gartner-says-worldwide-iot-security-spending-will-reach-1-point-5-billion-in-2018`. 2018 (cit. on p. 1).

[13] Philipp Wegner. *The 1,200 IoT companies that are creating the connected world of the future – IoT Startup Landscape 2021*. `https://iot-analytics.com/iot-startup-landscape/`. June 2021 (cit. on p. 2).

[14] Amazon Web Services Inc. *AWS IoT Core*. Amazon Web Services, `https://aws.amazon.com/iot-core/` (cit. on pp. 2, 9, 92).

[15] Google. *Google Cloud IoT Core*. `https://cloud.google.com/iot-core`. 2021 (cit. on pp. 2, 9, 92).

[16] Microsoft Azure. *Azure IoT Hub*. `https://azure.microsoft.com/en-us/services/iot-hub`. 2022 (cit. on pp. 2, 9, 99, 101, 108).

[17] S. Srinivasa, J. M. Pedersen, and E. Vasilomanolakis. “Open for hire: attack trends and misconfiguration pitfalls of IoT devices”. In: *ACM IMC*. 2021 (cit. on pp. 2, 25, 27).

[18] Arturs Lavrenovs and Gabor Visky. “Exploring features of HTTP responses for the classification of devices on the Internet”. In: *2019 27th Telecommunications Forum (TELFOR)*. 2019, pp. 1–4. DOI: `10.1109/TELFOR48224.2019.8971100` (cit. on pp. 2, 25).

[19] Oliver Gasser et al. “Security Implications of Publicly Reachable Building Automation Systems”. In: *2017 IEEE Security and Privacy Workshops (SPW)*. 2017, pp. 199–204. DOI: `10.1109/SPW.2017.13` (cit. on pp. 2, 25).

[20] L. Izhikevich, R. Teixeira, and Z. Durumeric. “LZR: Identifying Unexpected Internet Services”. In: *USENIX Security Symposium*. 2021 (cit. on pp. 2, 25, 27, 111, 117).

[21] Z. Durumeric et al. “A Search Engine Backed by Internet-Wide Scanning”. In: *ACM CCS*. 2015 (cit. on pp. 2, 25, 27, 103).

[22] Shodan. *Search Engine for the Internet of Everything*. `https://www.shodan.io/`. 2022 (cit. on pp. 2, 25, 27).

[23] *Open MQTT Report – Expanding the Hunt for Vulnerable IoT devices*. `https://www.shadowserver.org/news/open-mqtt-report-expanding-the-hunt-for-vulnerable-iot-devices/`. 2020 (cit. on pp. 2, 25, 27).

[24] A. Sivanathan, H. H. Gharakheili, and V. Sivaraman. "Can We Classify an IoT Device using TCP Port Scan?" In: *2018 IEEE International Conference on Information and Automation for Sustainability (ICIAfS)*. 2018, pp. 1–4 (cit. on pp. 2, 25).

[25] M. Nawrocki, T. C. Schmidt, and M Wählisch. "Uncovering Vulnerable Industrial Control Systems from the Internet Core". In: *IEEE/IFIP Network Operations and Management Symposium (NOMS)*. 2020 (cit. on pp. 2, 25).

[26] Ariana Mirian et al. "An Internet-wide view of ICS devices". In: *2016 14th Annual Conference on Privacy, Security and Trust (PST)*. 2016, pp. 96–103. DOI: `10.1109/PST.2016.7906943` (cit. on pp. 2, 25).

[27] D. Kumar and K. Shen and B. Case and D. Garg and G. Alperovich and D. Kuznetsov and R. Gupta and Z. Durumeric. "All Things Considered: An Analysis of IoT Devices on Home Networks". In: *USENIX Security Symposium*. 2019 (cit. on pp. 2, 25, 100).

[28] M. Hammad Mazhar and Z. Shafiq. "Characterizing Smart Home IoT Traffic in the Wild". In: *ACM/IEEE Conference on Internet of Things Design and Implementation*. 2020 (cit. on pp. 2, 3, 22, 23, 25–27, 64, 117).

[29] *TCPDUMP/LIBPCAP public repository*. `http://www.tcpdump.org/`. 2019 (cit. on pp. 3, 17).

[30] B. Claise. *Cisco Systems NetFlow Services Export Version 9*. RFC 3954 (Informational). Oct. 2004. URL: `http://www.ietf.org/rfc/rfc3954.txt` (cit. on pp. 3, 4, 26, 57, 59).

[31] B. Claise, B. Trammell, and P. Aitken. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. RFC 7011 (INTERNET STANDARD). Sept. 2013. URL: `http://www.ietf.org/rfc/rfc7011.txt` (cit. on pp. 3, 4, 16, 17, 26, 57, 59).

[32] Roman Kolcun et al. "Revisiting IoT Device Identification". In: *TMA*. 2021 (cit. on pp. 3, 26).

[33] Oliver Thompson, Anna Maria Mandalari, and Hamed Haddadi. "Rapid IoT Device Identification at the Edge". In: *Proceedings of the 2nd ACM International Workshop on Distributed Machine Learning*. DistributedML '21. Virtual Event, Germany: Association for Computing Machinery, 2021, pp. 22–28. ISBN: 9781450391344. DOI: `10.1145/3488659.3493777`. URL: `https://doi.org/10.1145/3488659.3493777` (cit. on pp. 3, 26).

[34] Arman Pashamokhtari et al. "Inferring Connected IoT Devices from IPFIX Records in Residential ISP Networks". In: *2021 IEEE 46th Conference on Local Computer Networks (LCN)*. IEEE. 2021, pp. 57–64 (cit. on pp. 3, 78).

[35] A. Sivanathan et al. "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics". In: *IEEE Transactions on Mobile Computing* 18.8 (2019) (cit. on pp. 3, 26).

[36] Rahmadi Trimananda et al. “Packet-level signatures for smart home devices”. In: *Network and Distributed Systems Security (NDSS) Symposium.* Vol. 2020. 2020 (cit. on pp. 3, 26).

[37] Arunan Sivanathan et al. “Characterizing and classifying IoT traffic in smart cities and campuses”. In: *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS).* 2017, pp. 559–564. DOI: `10.1109/INFCOMW.2017.8116438` (cit. on p. 3).

[38] H. Guo and J. Heidemann. “Detecting IoT Devices in the Internet”. In: *IEEE/ACM Transactions on Networking* (2020). [to appear] (cit. on pp. 3, 26, 78, 80).

[39] R. Perdisci et al. “IoTFinder: Efficient Large-Scale Identification of IoT Devices via Passive DNS Traffic Analysis”. In: *IEEE European Symposium of Security and Privacy.* 2020 (cit. on pp. 3, 26, 74, 80, 100).

[40] O. Alrawi et al. “SoK: Security Evaluation of Home-Based IoT Deployments”. In: *S&P.* 2019 (cit. on pp. 3, 27, 100, 101).

[41] O. Çetin et al. “Cleaning Up the Internet of Evil Things: Real-World Evidence on ISP and Consumer Efforts to Remove Mirai”. In: *NDSS.* 2019 (cit. on pp. 3, 79).

[42] Arman Noroozian et al. “Can ISPs help mitigate IoT malware? A longitudinal study of broadband ISP security efforts”. In: *2021 IEEE European Symposium on Security and Privacy (EuroS&P).* IEEE. 2021, pp. 337–352 (cit. on p. 3).

[43] Stephanie Borg Psaila. *Right to access the Internet: the countries and the laws that proclaim it.* `https://www.diplomacy.edu/blog/right-to-access-the-internet-countries-and-laws-proclaim-it/`. May 2011 (cit. on p. 11).

[44] Anja Feldmann et al. “The Lockdown Effect: Implications of the COVID-19 Pandemic on Internet Traffic”. In: IMC ’20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1–18. ISBN: 9781450381383. DOI: `10.1145/3419394.3423658`. URL: `https://doi.org/10.1145/3419394.3423658` (cit. on p. 11).

[45] James F. Kurose and Keith W. Ross. *Computer networking: A top-down approach, 6/E.* Pearson, Mar. 2012, pp. 32–35. ISBN: 9780132856201 (cit. on pp. 11–13).

[46] Amogh Dhamdhere and Constantine Dovrolis. “The Internet is Flat: Modeling the Transition from a Transit Hierarchy to a Peering Mesh”. In: *Proceedings of the 6th International COnference.* Co-NEXT ’10. Philadelphia, Pennsylvania: Association for Computing Machinery, 2010. ISBN: 9781450304481. DOI: `10.1145/1921168.1921196`. URL: `https://doi.org/10.1145/1921168.1921196` (cit. on p. 12).

[47] Amazon AWS. *Regions and Zones.* https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/usi regions-availability-zones.html. 2022 (cit. on pp. 13, 108, 109).

[48] Microsoft Azure. *Microsoft Azure.* `https://azure.microsoft.com/`. 2022 (cit. on p. 13).

[49] E. Nygren, R. K. Sitaraman, and J. Sun. “The Akamai Network: A Platform for High-performance Internet Applications”. In: *SIGOPS Oper. Syst. Rev.* 44.3 (2010) (cit. on p. 13).

[50] Cisco. *Introduction to Cisco IOS NetFlow - A Technical Overview.* `https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html`. 2012 (cit. on pp. 15, 17, 86, 111).

[51] B. Claise. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information.* RFC 5101 (Proposed Standard). Obsoleted by RFC 7011. Jan. 2008. URL: `http://www.ietf.org/rfc/rfc5101.txt` (cit. on p. 16).

[52] IUANA. *IP Flow Information Export (IPFIX) Entities.* `https://www.iana.org/assignments/ipfix/ipfix.xhtml`. 2022 (cit. on p. 16).

[53] R. Hofstede et al. “Flow Monitoring Explained: From Packet Captureto Data Analysis With NetFlow and IPFIX”. In: *IEEE Communications Surveys & Tutorials* 16.4 (2014) (cit. on p. 16).

[54] C. Cranor et al. “Gigascope: A Stream Database for Network Applications”. In: *ACM SIGMOD.* 2003 (cit. on pp. 17, 18, 34).

[55] D. Sarlis et al. “Datix: A system for scalable network analytics”. In: *ACM CCR* 45.5 (2015) (cit. on p. 17).

[56] G. Cormode et al. “Finding hierarchical heavy hitters in data streams”. In: *VLDB.* 2003 (cit. on p. 17).

[57] R. B. Basat et al. “Constant Time Updates in Hierarchical Heavy Hitters”. In: *ACM SIGCOMM.* 2017 (cit. on pp. 17–19, 38, 47, 49).

[58] A. R Curtis et al. “DevoFlow: scaling flow management for high-performance networks”. In: *ACM CCR.* Vol. 41. 4. 2011 (cit. on p. 17).

[59] A. Gupta et al. “Sonata: Query-driven Streaming Network Telemetry”. In: *ACM SIGCOMM.* 2018 (cit. on pp. 16–18, 54).

[60] S. Narayana et al. “Language-Directed Hardware Design forNetwork Performance Monitoring”. In: *ACM SIGCOMM.* 2017 (cit. on pp. 16–18).

[61] Q. Huang et al. “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. 17, 19).

[62] R. Ben Basat et al. “Optimal elephant flow detection”. In: *IEEE INFOCOM.* 2017 (cit. on p. 17).

[63] Y. Da et al. “dShark: A General, Easy to Program and Scalable Framework for Analyzing In-network Packet Traces”. In: *NSDI.* 2019 (cit. on pp. 17, 18).

[64] 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. 17).

[65] G. Cormode and M. Hadjieleftheriou. “Finding Frequent Items in Data Streams”. In: *VLDB*. 2008 (cit. on p. 17).

[66] G. Cormode and S. Muthukrishnan. “Space Efficient Mining of Multigraph Streams”. In: *PODS*. 2005 (cit. on p. 17).

[67] S. Narayana et al. “Compiling Path Queries”. In: *NSDI*. 2016 (cit. on p. 17).

[68] Y. Li et al. “FlowRadar: A Better NetFlow for Data Centers.” In: *Nsdi*. 2016, pp. 311–324 (cit. on p. 17).

[69] 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. 17).

[70] O. Tilmans et al. “Stroboscope: Declarative Network Monitoring on a Budget”. In: *NSDI* (2018) (cit. on pp. 16–18).

[71] T. Yang et al. “Elastic Sketch: Adaptive and Fast Network-wide Measurements”. In: *ACM SIGCOMM*. 2018 (cit. on pp. 17, 19, 47).

[72] M. Yu, L. Jose, and R. Miao. “Software Defined Traffic Measurement with OpenSketch”. In: *NSDI*. 2013 (cit. on pp. 17, 19).

[73] V. Bajpai and J. Schönwälder. “Network flow query language—Design, implementation, performance, and applications”. In: *IEEE TNSM* 14.1 (2016) (cit. on p. 17).

[74] 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. 17, 18).

[75] J. Shuyuan and D. S. Yeung. “A covariance analysis model for DDoS attack detection”. In: *IEEE ICC*. 2004 (cit. on p. 17).

[76] V. Sekar et al. “LADS: Large-scale Automated DDoS Detection System.” In: *USENIX Annual Technical Conference, General Track*. 2006, pp. 171–184 (cit. on p. 17).

[77] S. M. Mousavi and M. St-Hilaire. “Early detection of DDoS attacks against SDN controllers”. In: *ICNC*. 2015 (cit. on p. 17).

[78] 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. 17).

[79] R. Schweller et al. “Reversible sketches: enabling monitoring and analysis over high-speed data streams”. In: *IEEE/ACM Trans. Networking* 15.5 (2007) (cit. on p. 17).

[80] 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. 17).

[81] C. Graham and S. Muthukrishnan. “What’s new: Finding significant differences in network data streams”. In: *IEEE INFOCOM*. 2004 (cit. on p. 17).

[82] P. Tammana, R. Agarwal, and M. Lee. “Simplifying datacenter network debugging with pathdump”. In: *ACM OSDI*. 2016 (cit. on p. 17).

[83] P. Tammana, R. Agarwal, and M. Lee. “Distributed Network Monitoring and Debugging with SwitchPointer”. In: *NSDI*. 2018 (cit. on p. 17).

[84] B. Arzani et al. “007: Democratically Finding the Cause of Packet Drops”. In: *NSDI*. 2018 (cit. on p. 17).

[85] C. Estan et al. “Building a better NetFlow”. In: *ACM SIGCOMM*. 2004 (cit. on pp. 17, 18).

[86] *InMon – bhuyan2015towards*. `http://sflow.org/`. 2019 (cit. on p. 17).

[87] 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. 17, 18).

[88] C. Labovitz et al. “Internet Inter-Domain Traffic”. In: *ACM SIGCOMM*. 2010 (cit. on p. 17).

[89] R. Caceres et al. “Measurement and analysis of IP network usage and behavior”. In: *IEEE Communications Magazine* 38.5 (2000) (cit. on p. 17).

[90] 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. 18).

[91] A. Wundsam et al. “OFRewind: Enabling Record and Replay Troubleshooting for Networks”. In: *Usenix ATC*. 2011 (cit. on p. 18).

[92] A. Wundsam et al. “Network troubleshooting with mirror vnets”. In: *GLOBECOM Workshops*. 2010 (cit. on p. 18).

[93] Ross Teixeira et al. “Packetscope: Monitoring the packet lifecycle inside a switch”. In: *Proceedings of the Symposium on SDN Research*. 2020, pp. 76–82 (cit. on p. 18).

[94] Muhammad Tirmazi et al. “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. 18).

[95] Damu Ding et al. “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. 18).

[96] Ran Ben Basat et al. “Designing Heavy-Hitter Detection Algorithms for Programmable Switches”. In: *IEEE/ACM Transactions on Networking* (2020) (cit. on p. 18).

[97] S. Pontarelli et al. “Flowblaze: Stateful packet processing in hardware”. In: *NSDI*. 2019 (cit. on p. 18).

[98] M. Zhang et al. “Poseidon: Mitigating volumetric ddos attacks with programmable switches”. In: *NDSS*. 2020 (cit. on p. 18).

[99] M. Yu. “Network telemetry: towards a top-down approach”. In: *ACM CCR* 49.1 (2019) (cit. on p. 18).

[100] Y. Lee and Y. Lee. “Toward Scalable Internet Traffic Measurement and Analysis with Hadoop”. In: *ACM CCR* 43.1 (2013) (cit. on p. 18).

[101] Yandex. *ClickHouse – open source distributed column-oriented DBMS*. https://clickhouse.yandex/. 2018 (cit. on pp. 18, 54).

[102] A. Vulimiri et al. “Global analytics in the face of bandwidth and regulatory constraints”. In: *NSDI*. 2015 (cit. on p. 18).

[103] A. Vulimir et al. “WANalytics: Analytics for a Geo-Distributed Data-Intensive Worl”. In: *CIDR*. 2015 (cit. on p. 18).

[104] R. Viswanathan, G. Ananthanarayanan, and A. Akella. “CLARINET: WAN-Aware Optimization for Analytics Queries”. In: *NSDI*. 2016 (cit. on p. 18).

[105] K. Hsieh et al. “Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds”. In: *NSDI*. 2017 (cit. on p. 18).

[106] Yuzhen Huang et al. “Yugong: Geo-Distributed data and job placement at scale”. In: *Proceedings of the VLDB Endowment* 12.12 (2019), pp. 2155–2169 (cit. on p. 18).

[107] Alessandro D’Alconzo et al. “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. 18).

[108] Ran Ben Basat et al. “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. 18).

[109] C. Estan and G. Varghese. “New Directions in Traffic Measurement and Accounting”. In: *ACM SIGCOMM*. 2002 (cit. on p. 18).

[110] Rob Harrison et al. “Carpe Elephants: Seize the Global Heavy Hitters”. In: *Proceedings of the Workshop on Secure Programmable Network Infrastructure*. 2020, pp. 15–21 (cit. on p. 18).

[111] G. Cormode et al. “Diamond in the Rough: Finding Hierarchical Heavy Hitters in Multi-Dimensional Data”. In: *ACM SIGMOD*. 2004 (cit. on pp. 18, 36–38).

[112] M. Mitzenmacher, T. Steinke, and J. Thaler. “Hierarchical Heavy Hitters with the Space Saving Algorithm”. In: *ALENEX*. 2012 (cit. on pp. 18, 38, 49).

[113] 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. 19).

[114] Nikita Ivkin et al. “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. 19).

[115] Z. Liu et al. “One sketch to rule them all: Rethinking network flow monitoring with univmon”. In: *ACM SIGCOMM*. 2016 (cit. on p. 19).

[116] Theophilus Wellem et al. “A flexible sketch-based network traffic monitoring infrastructure”. In: *IEEE Access* 7 (2019) (cit. on p. 19).

[117] Haibo Wang et al. “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. 19).

[118] ITU. *Security framework for the Internet of things based on the gateway model.* https://handle.itu.int/11.1002/1000/13607. 2018 (cit. on p. 21).

[119] Amazon AWS. *Open MQTT Report – Expanding the Hunt for Vulnerable IoT devices.* https://aws.amazon.com/what-is/iot/ (cit. on p. 21).

[120] IEEE. *Towards a Definition of the Internet of Things (IoT).* https://iot.ieee.org/definition.html. 2015 (cit. on p. 21).

[121] Mehwish Akram et al. *Securing Web Transactions: TLS Server Certificate Management.* Tech. rep. National Institute of Standards and Technology, 2020 (cit. on p. 22).

[122] D. Mills et al. *Network Time Protocol Version 4: Protocol and Algorithms Specification.* RFC 5905 (Proposed Standard). Updated by RFCs 7822, 8573, 9109. June 2010. URL: http://www.ietf.org/rfc/rfc5905.txt (cit. on p. 24).

[123] Eclipse Foundation. *2021 IoT & Edge Developer Survey Report.* https://outreach.eclipse.foundation/iot-edge-developer-2021. Jan. 2022 (cit. on p. 24).

[124] OASIS Foundation. *MQTT Specifications.* https://mqtt.org/mqtt-specification/. 2022 (cit. on p. 24).

[125] Philipp Richter et al. "Distilling the internet's application mix from packet-sampled traffic". In: *International Conference on Passive and Active Network Measurement.* Springer. 2015, pp. 179–192 (cit. on p. 24).

[126] Z. Shelby, K. Hartke, and C. Bormann. *The Constrained Application Protocol (CoAP).* RFC 7252 (Proposed Standard). Updated by RFCs 7959, 8613, 8974, 9175. June 2014. URL: http://www.ietf.org/rfc/rfc7252.txt (cit. on p. 25).

[127] Poonam Yadav et al. "Position Paper: A Systematic Framework for Categorising IoT Device Fingerprinting Mechanisms". In: *Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things.* AIChallengeIoT '20. Virtual Event, Japan: Association for Computing Machinery, 2020, pp. 62–68. ISBN: 9781450381345. DOI: 10.1145/3417313.3429384. URL: https://doi.org/10.1145/3417313.3429384 (cit. on p. 25).

[128] Xuan Feng et al. "Acquisitional Rule-Based Engine for Discovering Internet-of-Thing Devices". In: *Proceedings of the 27th USENIX Conference on Security Symposium.* SEC'18. Baltimore, MD, USA: USENIX Association, 2018, pp. 327–341. ISBN: 9781931971461 (cit. on p. 25).

[129] A. Sivanathan, H. H. Gharakheili, and V. Sivaraman. "Inferring IoT Device Types from Network Behavior Using Unsupervised Clustering". In: *IEEE Conference on Local Computer Networks (LCN).* 2019 (cit. on p. 26).

[130] S. Marchal et al. "AUDI: Towards Autonomous IoT Device-Type Identification using Periodic Communication". In: *IEEE Journal on Sel. Areas in Comm.* 37.6 (2019) (cit. on p. 26).

[131] Bharat Atul Desai et al. “A feature-ranking framework for IoT device classification”. In: *2019 11th International Conference on Communication Systems and Networks (COMSNETS)*. 2019, pp. 64–71. DOI: `10.1109/COMSNETS.2019.8711210` (cit. on p. 26).

[132] N. Apthorpe, D. Reisman, and N. Feamster. “A Smart Home is No Castle: Privacy Vulnerabilities of Encrypted IoT Traffic”. In: *Data and Algorithmic Transparency Workshop* (2016) (cit. on p. 26).

[133] J. Ren et al. “Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach”. In: *ACM IMC*. 2019 (cit. on pp. 26, 27, 57, 62, 64).

[134] D. Y. Huang et al. “IoTInspector: Crowdsourcing Labeled Network Traffic from Smart Home Devices at Scale”. In: *ACM IMWUT / Ubicomp*. 2020 (cit. on p. 26).

[135] G. Hu and K. Fukuda. “Toward Detecting IoT Device Traffic in Transit Networks”. In: *International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. 2020 (cit. on p. 26).

[136] Z. Durumeric, E. Wustrow, and J. A. Halderman. “ZMap: Fast Internet-Wide Scanning and its Security Applications”. In: *USENIX Security Symposium*. 2013 (cit. on pp. 27, 29).

[137] X. He et al. “Fingerprinting Mainstream IoT Platforms Using Traffic Analysis”. In: *IEEE Internet of Things Journal* 9.3 (2022), pp. 2083–2093 (cit. on pp. 27, 100, 101).

[138] *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing*. RFC 2827. 2000 (cit. on pp. 28, 112).

[139] C. Partridge and M. Allman. “Ethical Considerations in Network Measurement Papers”. In: *Communications of the ACM* (2016) (cit. on p. 29).

[140] 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. 29).

[141] *KrebsOnSecurity Hit With Record DDoS*. `https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos` (cit. on p. 31).

[142] *Dyn Analysis Summary Of Friday October 21 Attack*. `https://dyn.com/blog/dyn-analysis- summary-of-friday-october-21-attack/`. Oct. 2016 (cit. on p. 31).

[143] *Flowyager in Github*. `https://github.com/saidjawad/Flowyager` (cit. on p. 31).

[144] N. Duffield, C. Lund, and M. Thorup. “Estimating Flow Distributions from Sampled Flow Statistics”. In: *ACM SIGCOMM*. 2003 (cit. on p. 32).

[145] T. Repantis et al. “Scaling a Monitoring Infrastructure for the Akamai Network”. In: *SIGOPS Oper. Syst. Rev.* 44.3 (2010) (cit. on p. 34).

[146] J. Cohen et al. "Keeping Track of 70,000+ Servers: The Akamai Query System". In: *USENIX LISA*. 2010 (cit. on p. 34).

[147] J. Wallerich et al. "A Methodology for Studying Persistency Aspects of Internet Flows". In: *ACM CCR* 35.2 (Apr. 2005) (cit. on p. 36).

[148] Y. Zhang et al. "On the characteristics and origins of internet flow rates". In: *ACM CCR*. Vol. 32. 4. 2002 (cit. on p. 36).

[149] L. Breslau et al. "Web caching and Zipf-like distributions: Evidence and implications". In: *IEEE INFOCOM*. 1999 (cit. on p. 36).

[150] G. Cormode et al. "Finding Hierarchical Heavy Hitters in Streaming Data". In: *ACM Trans. Knowl. Discov. Data* 1.4 (2008) (cit. on pp. 36–38).

[151] *mongoDB: The database for modern applications.* `https://www.mongodb.com/`. 2019 (cit. on p. 43).

[152] *Apache Thrift.* `https://thrift.apache.org/`. 2019 (cit. on p. 43).

[153] *ANTLR (ANother Tool for Language Recognition).* `https://www.antlr.org/`. 2019 (cit. on p. 43).

[154] RStudio, Inc. *Easy Web applications in R.* 2013 (cit. on p. 43).

[155] *MAWI Working Group Traffic Archive.* `http://mawi.wide.ad.jp/mawi/`. 2018 (cit. on p. 45).

[156] B. B. Ran. *Implementation of the Constant Time Updates in Hierarchical Heavy Hitters paper, ACM SIGCOMM 2017.* `https://github.com/ranbenbasat/RHHH`. 2018 (cit. on p. 49).

[157] CAIDA. *The CAIDA UCSD Passive Monitor: Equinix-Chicago - 2016-02-18.* `https://www.caida.org/data/monitors/passive-equinix-chicago.xml`. 2018 (cit. on p. 49).

[158] 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. 54).

[159] G. Carl et al. "Denial-of-service attack-detection techniques". In: *IEEE Internet computing* 10.1 (2006), pp. 82–89 (cit. on p. 54).

[160] 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. 54).

[161] K. Lee et al. "DDoS attack detection method using cluster analysis". In: *Expert systems with applications* 34.3 (2008), pp. 1659–1665 (cit. on p. 54).

[162] J. Czyz et al. "Taming the 800 Pound Gorilla: The Rise and Decline of NTP DDoS Attacks". In: *ACM IMC*. 2014 (cit. on pp. 54, 55).

[163] M. Jonker et al. "Millions of Targets Under Attack: A Macroscopic Characterization of the DoS Ecosystem". In: *ACM IMC*. 2017 (cit. on pp. 54, 55).

[164] C. Dietzel et al. "Stellar: network attack mitigation using advanced blackholing". In: *ACM CoNEXT*. 2018 (cit. on pp. 54, 55).

[165] M. Jonker et al. “Measuring the Adoption of DDoS Protection Services”. In: *ACM IMC*. 2016 (cit. on p. 54).

[166] *US-Cert: Alert (TA14-017A), UDP-Based Amplification Attacks*. `https://www.us-cert.gov/ncas/alerts/TA14-017A`. 2019 (cit. on p. 55).

[167] 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. 55).

[168] C. Rossow. “Amplification Hell: Revisiting Network Protocols for DDoS Abuse”. In: *NDSS* (2014) (cit. on p. 55).

[169] Amazon. *Alexa Voice Service Endpoints (accessed 2019-11)*. `https://developer.amazon.com/en-US/docs/alexa/alexa-voice-service/api-overview.html#endpoints` (cit. on pp. 57, 64, 69).

[170] L. F. DeKoven et al. “Measuring Security Practices and How They Impact Security”. In: *ACM IMC*. 2019 (cit. on pp. 59, 74).

[171] P. Patel et al. “On the Effectiveness of Random Testing for Android: Or How I Learned to Stop Worrying and Love the Monkey”. In: *Proceedings of the 13th International Workshop on Automation of Software Test*. 2018 (cit. on p. 61).

[172] *Farsight Security DNSDB*. `https://www.dnsdb.info/` (cit. on pp. 67, 103).

[173] Z. Durumeric et al. “A Search Engine Backed by Internet-Wide Scanning”. In: *ACM CCS*. 2015 (cit. on pp. 67, 68).

[174] C. Iordanou et al. “Tracing cross border web tracking”. In: *ACM IMC*. 2018 (cit. on pp. 67, 105).

[175] F. Weimer. “Passive DNS Replication”. In: *17th Annual FIRST Conference*. 2005 (cit. on pp. 67, 103).

[176] Amazon AWS. *What is Amazon VPC? (accessed 2019-11)*. `https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html` (cit. on p. 68).

[177] Amazon AWS. *Public IPv4 addresses and external DNS hostnames (accessed 2019-11)*. `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-instance-addressing.html#concepts-public-addresses` (cit. on p. 68).

[178] Microsoft. *Public IP addresses (accessed 2019-11)*. `https://docs.microsoft.com/en-us/azure/virtual-network/virtual-network-ip-addresses-overview-arm#public-ip-addresses` (cit. on p. 68).

[179] Tuya Inc. *Tuya Platform and Services (accessed 2019-12)*. `https://www.tuya.com/platform` (cit. on p. 69).

[180] Electric Imp. *Electric imp Platform (accessed 2019-12)*. `https://www.electricimp.com/platform/how-it-works/` (cit. on p. 69).

[181] Amazon. *AWS IoT Platform (accessed 2019-12)*. `https://aws.amazon.com/iot/` (cit. on pp. 69, 99, 101, 108).

[182] AuraK, Samsung Community Moderator. *Backgroundverbindungen (auch Standby), Datenschutz - in German(accessed 2019-11).* `https://eu.community.samsung.com/t5/TV/Backgroundverbindungen-auch-Standby-Datenschutz/m-p/625473/highlight/true\#M24445`. July 2018 (cit. on pp. 69, 70).

[183] Bitkom e.V. *Zukunft der Consumer Technology – 2019 - in German (accessed 2019-11).* `https://www.bitkom.org/sites/default/files/2019-09/190903_ct_studie_2019_online.pdf`. 2019 (cit. on p. 74).

[184] IDC. *Google Overtakes Amazon to Lead the European Smart Home Market in 1Q19, says IDC (accessed 2019-11).* `https://www.idc.com/getdoc.jsp?containerId=prEUR145337319`. 2019 (cit. on p. 74).

[185] Deutsche Welle. *Voice Assistants on the rise in Germany (accessed 2019-11).* `https://www.dw.com/en/voice-assistants-on-the-rise-in-germany/a-45269599` (cit. on p. 74).

[186] A. H. Rasti et al. "Eyeball ASes: from Geography to Connectivity". In: *ACM IMC.* 2010 (cit. on p. 78).

[187] M. Antonakakis et al. "Understanding the Mirai Botnet". In: *USENIX Security Symposium.* 2017 (cit. on p. 79).

[188] A. M. Mandalari et al. "Towards Automatic Identification and Blocking of Non-Critical IoT Traffic Destinations". In: *IEEE S & P Workshop on Technology and Consumer Protection.* 2020 (cit. on p. 79).

[189] Google. *DNS-over-TLS.* `https://developers.google.com/speed/public-dns/docs/dns-over-tls`. 2020 (cit. on p. 80).

[190] F. Chen, R. K. Sitaraman, and M. Torres. "End-User Mapping: Next Generation Request Routing for Content Delivery". In: *ACM SIGCOMM.* 2015 (cit. on p. 80).

[191] S. Strowes. *IPv6 Adoption in 2021.* `https://labs.ripe.net/author/stephen_strowes/ipv6-adoption-in-2021/`. 2021 (cit. on p. 83).

[192] G. Huston. *IPv6 and the Internet of Things.* `https://blog.apnic.net/2016/04/13/ipv6-internet-things/`. 2016 (cit. on p. 83).

[193] T. Mrugalski et al. *Dynamic Host Configuration Protocol for IPv6 (DHCPv6).* RFC 8415 (Proposed Standard). Nov. 2018. URL: `http://www.ietf.org/rfc/rfc8415.txt` (cit. on pp. 83, 84).

[194] T. Narten, R. Draves, and S. Krishnan. *Privacy Extensions for Stateless Address Autoconfiguration in IPv6.* RFC 4941 (Draft Standard). Obsoleted by RFC 8981. Sept. 2007. URL: `http://www.ietf.org/rfc/rfc4941.txt` (cit. on pp. 83, 84, 95).

[195] F. Gont et al. *Temporary Address Extensions for Stateless Address Autoconfiguration in IPv6.* RFC 8981 (Proposed Standard). Feb. 2021. URL: `http://www.ietf.org/rfc/rfc8981.txt` (cit. on pp. 83, 95).

[196] E. C. Rye, R. Beverly, and kc claffy. "Follow the Scent: Defeating IPv6 Prefix Rotation Privacy". In: *ACM IMC.* 2021 (cit. on pp. 83, 89).

[197] P. Gigis et al. “Seven Years in the Life of Hypergiants’ Off-Nets”. In: *ACM SIGCOMM*. 2021 (cit. on pp. 83, 100, 103).

[198] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460 (Draft Standard). Obsoleted by RFC 8200, updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112. Dec. 1998. URL: `http://www.ietf.org/rfc/rfc2460.txt` (cit. on p. 84).

[199] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 8200 (INTERNET STANDARD). July 2017. URL: `http://www.ietf.org/rfc/rfc8200.txt` (cit. on p. 84).

[200] E. Karpilovsky et al. “Quantifying the extent of IPv6 deployment”. In: *PAM*. 2009 (cit. on p. 84).

[201] Google. *IPv6 Adoption*. `https://www.google.com/intl/en/ipv6/statistics.html` (cit. on p. 84).

[202] F. Gont and T. Chown. *Network Reconnaissance in IPv6 Networks*. RFC 7707 (Informational). Mar. 2016. URL: `http://www.ietf.org/rfc/rfc7707.txt` (cit. on p. 84).

[203] R. Droms. *Dynamic Host Configuration Protocol*. RFC 2131 (Draft Standard). Updated by RFCs 3396, 4361, 5494, 6842. Mar. 1997. URL: `http://www.ietf.org/rfc/rfc2131.txt` (cit. on p. 84).

[204] S. Thomson, T. Narten, and T. Jinmei. *IPv6 Stateless Address Autoconfiguration*. RFC 4862 (Draft Standard). Updated by RFC 7527. Sept. 2007. URL: `http://www.ietf.org/rfc/rfc4862.txt` (cit. on p. 84).

[205] IEEE Standards Association. *Guidelines for Use of Extended Unique Identifier (EUI), Organizationally Unique Identifier (OUI), and Company ID (CID)*. `https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/tutorials/eui.pdf`. 2018 (cit. on p. 84).

[206] E. C. Rye, J. Martin, and R. Beverly. “EUI-64 Considered Harmful”. In: *arXiv preprint arXiv:1902.08968* (2019) (cit. on p. 84).

[207] O. Gasser et al. “Scanning the IPv6 Internet: Towards a Comprehensive Hitlist”. In: *TMA*. 2016 (cit. on p. 84).

[208] O. Gasser et al. “Clusters in the Expanse: Understanding and Unbiasing IPv6 Hitlists”. In: *ACM IMC*. 2018 (cit. on pp. 84, 103).

[209] E. C. Rye and R. Beverly. “Discovering the IPv6 network periphery”. In: *arXiv preprint arXiv:2001.08684* (2020) (cit. on p. 84).

[210] R. Beverly et al. “In the IP of the beholder: Strategies for active IPv6 topology discovery”. In: *ACM IMC*. 2018 (cit. on p. 84).

[211] J. P. Rohrer, B. LaFever, and R. Beverly. “Empirical study of router IPv6 interface address distributions”. In: *IEEE Internet Computing* 20.4 () (cit. on p. 84).

[212] S. D. Strowes. “Bootstrapping Active IPv6 Measurement with IPv4 and Public DNS”. In: *arXiv preprint arXiv:1710.08536* (2017) (cit. on p. 84).

[213] T. Fiebig et al. “Something from Nothing (There): Collecting Global IPv6 Datasets from DNS”. In: *PAM*. 2017 (cit. on p. 84).

[214] T. Fiebig et al. “In rDNS We Trust: Revisiting a Common Data-Source’s Reliability”. In: *PAM*. 2018 (cit. on p. 84).

[215] K. Borgolte et al. “Enumerating Active IPv6 Hosts for Large-scale Security Scans via DNSSEC-signed Reverse Zones”. In: *IEEE Symposium on Security and Privacy*. 2018 (cit. on p. 84).

[216] J. Ullrich et al. “On reconnaissance with IPv6: a pattern-based scanning approach”. In: *ARES*. 2015 (cit. on p. 84).

[217] P. Foremski, D. Plonka, and A. Berger. “Entropy/IP: Uncovering Structure in IPv6 Addresses”. In: *ACM IMC*. 2016 (cit. on p. 84).

[218] A. Murdock et al. “Target Generation for Internet-wide IPv6 Scanning”. In: *ACM IMC*. 2017 (cit. on p. 84).

[219] K. Fukuda and J. Heidemann. “Who Knocks at the IPv6 Door? Detecting IPv6 Scanning”. In: *ACM IMC*. 2018 (cit. on p. 84).

[220] Z. Liu et al. “6Tree: Efficient dynamic discovery of active addresses in the IPv6 address space”. In: *Computer Networks* 155 (2019) (cit. on p. 84).

[221] V. Bajpai and J. Schönwälder. “A Longitudinal View of Dual-Stacked Websites—Failures, Latency and Happy Eyeballs”. In: *IEEE/ACM Transactions on Networking* 27.2 (2019) (cit. on p. 84).

[222] R. Almeida et al. “Classification of Load Balancing in the Internet”. In: *IEEE INFOCOM*. 2020 (cit. on p. 84).

[223] F. Li and D. Freeman. “Towards A User-Level Understanding of IPv6 Behavior”. In: *ACM IMC*. 2020 (cit. on p. 84).

[224] R. Padmanabhan et al. “DynamIPs: Analyzing address assignment practices in IPv4 and IPv6”. In: *ACM CoNEXT*. 2020 (cit. on p. 84).

[225] B. Hou et al. “6Hit: A Reinforcement Learning-based Approach to Target Generation for Internet-wide IPv6 Scanning”. In: *IEEE INFOCOM*. 2021 (cit. on p. 84).

[226] T. Cui et al. “6GAN: IPv6 Multi-Pattern Target Generation via Generative Adversarial Nets with Reinforcement Learning”. In: *IEEE INFOCOM*. 2021 (cit. on p. 84).

[227] X. Li et al. “Fast IPv6 Network Periphery Discovery and Security Implications”. In: *IEEE/IFIP DSN*. 2021 (cit. on p. 84).

[228] G. Zheng, X. Xu, and C. Wang. “An Effective Target Address Generation Method for IPv6 Address Scan”. In: *IEEE ICCC*. 2020 (cit. on p. 84).

[229] T. Bruns. “Network Reconnaissance in IPv6-based Residential Broadband Networks”. In: *arXiv preprint arXiv:2012.10652* (2020) (cit. on pp. 84, 95).

[230] Institute of Electrical and Electronics Engineers (IEEE). *Organizationally Unique Identifier (OUI) MAC Address Registry*. `http://standards-oui.ieee.org/oui/oui.txt` (cit. on pp. 85, 89).

[231] Apple Inc. *IPv6 security.* Apple Platform Security, https://support.apple.com/guide/security/ipv6-security-seccb625dcd9/web (cit. on p. 89).

[232] S. J. Saidi et al. “A Haystack Full of Needles: Scalable Detection of IoT Devices in the Wild”. In: *ACM IMC.* 2020 (cit. on pp. 92, 100, 105).

[233] A. Banks and R. Gupta. *MQTT Version 3.1.1.* MQTT Version 3.1.1, https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (cit. on p. 92).

[234] Google. *Android Enterprise Network Requirements.* https://support.google.com/work/android/answer/10513641?hl=en. 2021 (cit. on p. 93).

[235] T. Böttger et al. “A Hypergiant’s View of the Internet”. In: *ACM SIGCOMM CCR* 47.1 (2017) (cit. on p. 94).

[236] European Commission. *Internal Market, Industry, Entrepreneurship and SMEs - CE Marking.* https://ec.europa.eu/growth/single-market/ce-marking_en. 2021 (cit. on p. 96).

[237] G. Van de Velde et al. *Local Network Protection for IPv6.* RFC 4864 (Informational). May 2007. URL: http://www.ietf.org/rfc/rfc4864.txt (cit. on p. 96).

[238] Google. *AI and machine learning products.* https://cloud.google.com/products/ai. 2022 (cit. on p. 99).

[239] Amazon AWS. *Machine Learning on AWS.* https://aws.amazon.com/machine-learning/. 2022 (cit. on p. 99).

[240] G. Kappel et al. “Internet of Production: Entering Phase Two of Industry 4.0”. In: *Comm. of the ACM* (2022) (cit. on p. 99).

[241] Amazon AWS. *Amazon Personalize.* https://aws.amazon.com/personalize/. 2022 (cit. on p. 99).

[242] Google. *Google Cloud IoT solutions.* https://cloud.google.com/solutions/iot. 2022 (cit. on pp. 99, 101, 108).

[243] W. Zhou et al. “Discovering and Understanding the Security Hazards in the Interactions between IoT Devices, Mobile Apps, and Clouds on Smart Home Platforms”. In: *Usenix Security.* 2019 (cit. on pp. 100, 101).

[244] T. Arnold et al. “Cloud Provider Connectivity in the Flat Internet”. In: *ACM IMC.* 2020 (cit. on p. 100).

[245] T. K. Dang et al. “Cloudy with a Chance of Short RTTsAnalyzing Cloud Connectivity in the Internet”. In: *ACM IMC.* 2021 (cit. on p. 100).

[246] Bosch. *Bosch IoT Hub: protocol-adapters.* https://docs.bosch-iot-suite.com/hub/how-to/protocol-adapters/. 2022 (cit. on pp. 101, 102, 108).

[247] Fujitsu. *Fujitsu IoT.* https://iot-docs.jp-east-1.paas.cloud.global.fujitsu.com/en/manual/v7/apireference_en.pdf. 2022 (cit. on pp. 101, 108).

[248] Siemens. *MindSphere Documentation.* https://siemens.mindsphere.io/en/docs/documentation-overview. 2022 (cit. on pp. 101, 108).

[249] Huawei. *IoTDA*. https://www.huaweicloud.com/product/iothub.html. 2022 (cit. on pp. 101, 108).

[250] IBM. *IoT Solutions*. https://www.ibm.com/cloud/internet-of-things. 2022 (cit. on pp. 101, 108).

[251] Oracle. *Oracle Internet of Things Cloud Service*. https://docs.oracle.com/en/cloud/paas/iot-cloud/index.html. 2022 (cit. on pp. 101, 108).

[252] SAP. *SAP Internet of Things*. https://www.sap.com/products/iot-data-services.html. 2022 (cit. on pp. 101, 108).

[253] Alibaba. *IoT Platform*. https://www.alibabacloud.com/product/iot. 2022 (cit. on pp. 101, 108).

[254] Baidu. *IoT Platform*. https://intl.cloud.baidu.com/product/iot.html. 2022 (cit. on pp. 101, 108).

[255] B. Yeganeh et al. "How Cloud Traffic Goes Hiding:A Study of Amazon's Peering Fabric". In: *ACM IMC*. 2019 (cit. on p. 101).

[256] IoT Analytics. *2020 List of IoT Platforms Companies*. https://iot-analytics.com/product/list-of-iot-platform-companies. 2021 (cit. on p. 102).

[257] Alibaba. *Protocols for connecting devices*. https://partners-intl.aliyun.com/help/en/iot-platform/latest/protocols-for-connecting-devices. 2022 (cit. on pp. 102, 108).

[258] Tencent. *Device Connection Regions*. https://intl.cloud.tencent.com/document/product/1105/42712. 2022 (cit. on pp. 102, 108).

[259] F. Cangialosi et al. "Measurement and analysis of private key sharing in the https ecosystem". In: *ACM CCS*. 2016 (cit. on p. 103).

[260] T. Chung et al. "Measuring and applying invalid SSL certificates: The silent majority". In: *ACM IMC*. 2016 (cit. on p. 103).

[261] Z. Durmeric. *ZGrab2*. https://github.com/zmap/zgrab2. 2018 (cit. on p. 103).

[262] Alibaba. *Access IoT Platform by using HTTP*. https://www.alibabacloud.com/help/en/iot-platform/latest/access-iot-platform-by-using-http. 2022 (cit. on p. 106).

[263] Amazon. *Elastic IP addresses*. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html. 2022 (cit. on p. 108).

[264] Alibaba. *IPv6-based MQTT connections*. https://partners-intl.aliyun.com/help/en/iot-platform/latest/ipv6-based-mqtt-connections. 2022 (cit. on p. 108).

[265] Amazon AWS. *Device communication protocols*. https://docs.aws.amazon.com/iot/latest/developerguide/protocols.html. 2022 (cit. on p. 108).

[266] Baidu. *IoT Core*. https://cloud.baidu.com/doc/IOT/index.html. 2022 (cit. on p. 108).

[267] Baidu. *Region - IoT services related product area information.* https://intl.cloud.baidu.com/-doc/Reference/s/2jwvz23xx-en. 2022 (cit. on pp. 108, 109).

[268] Cisco. *Cisco Kinetic.* https://developer.cisco.com/site/kinetic/. 2022 (cit. on p. 108).

[269] Cisco Kinetic. *IoT Platform.* https://developer.cisco.com/docs/GMM/#!requirements/requirements. 2022 (cit. on p. 108).

[270] Google. *Publishing over the MQTT: MQTT Server.* https://cloud.google.com/iot/docs/how-tos. 2022 (cit. on pp. 108, 111).

[271] IBM. *Connect your device to Watson IoT Platform.* https://cloud.ibm.com/docs/IoT/index.html#step2. 2022 (cit. on p. 108).

[272] Microsoft Azure. *IoT Hub IP addresses.* https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-understand-ip-address. 2022 (cit. on p. 108).

[273] Oracle. *About the IoT Connectivity Protocols.* https://docs.oracle.com/en/cloud/paas/iot-cloud/develop/iot-connectivity-protocols.html. 2022 (cit. on p. 108).

[274] PTC. *ThingWorx IIoT Solutions Platform.* https://www.ptc.com/en/products/thingworx. 2022 (cit. on p. 108).

[275] SAP. *SAP IoT Device Connectivity.* https://help.sap.com/docs/SAP_IoT/226d46a15bb245b7bf8126604bd6f0fb/9c7273450a874772ad2db007ce212a79.html?version=2112b. 2022 (cit. on p. 108).

[276] Siemens. *MindConnect API.* https://documentation.mindsphere.io/MindSphere/apis/connectivity-mindconnect/api-mindconnect-overview.html#access. 2022 (cit. on p. 108).

[277] Sierra Wireless. *Getting Started with AirVantage Platform.* https://source.sierrawireless.com/airvantage/av/howto/gettingstarted/. 2022 (cit. on p. 108).

[278] Sierra Wireless. *Introduction to MQTT.* https://source.sierrawireless.com/airvantage/av/reference/hardware/protocols/mqtt/. 2022 (cit. on p. 108).

[279] Sierra Wireless. *How to configure my infrastructure when devices are in a private APN?* https://source.sierrawireless.com/airvantage/av/reference/register/howtos/configureInfrastructureForPrivateAPN/. 2022 (cit. on p. 108).

[280] Tencent. *IoT Hub.* https://intl.cloud.tencent.com/document/product/1105. 2022 (cit. on p. 108).

[281] S. J. Saidi et al. *Deep Dive into the IoT Backend Ecosystem artifacts.* https://github.com/saidjawad/iot-backend. 2022 (cit. on p. 108).

[282] Google. *Google Cloud Locations.* https://cloud.google.com/about/locations. 2022 (cit. on p. 109).

[283] Alibaba. *Regions and zones.* https://www.alibabacloud.com/help/en/basics-for-beginners/latest/regions-and-zones. 2022 (cit. on p. 109).

[284] Huawei. *Huawei Cloud Regions and Endpoints.* https://developer.huaweicloud.com/intl/en-us/endpoint. 2022 (cit. on p. 109).

[285] *Routeviews Project – University of Oregon.* http://www.routeviews.org/ (cit. on p. 110).

[286] Amazon AWS. *AWS Global Accelerator.* https://aws.amazon.com/global-accelerator/. 2022 (cit. on p. 110).

[287] P. Richter and A. Berger. "Scanning the Scanners: Sensing the Internet from a Massively Distributed Network Telescope". In: *ACM IMC.* 2019 (cit. on p. 112).

[288] Apache ActiveMQ. *Apache ActiveMQ Artemis.* https://activemq.apache.org/components/artemis/documentation/1.4.0/. 2022 (cit. on p. 117).

[289] J. Varmarken et al. "The TV is Smart and Full of Trackers:Measuring Smart TV Advertising and Tracking". In: 2020 (cit. on p. 117).

[290] J. Varmarken et al. "FingerprinTV: Fingerprinting Smart TV Apps". In: 2022 (cit. on p. 117).

[291] AWS. *Summary of the AWS Service Event in the Northern Virginia (US-EAST-1) Region.* https://aws.amazon.com/message/12721/. 2021 (cit. on p. 121).

[292] S. Soper and J. Gillum. *Amazon says software problem was at root of huge Internet outage this week.* https://fortune.com/2021/12/10/amazon-software-problem-cloud-outage-cause/. 2021 (cit. on p. 121).

[293] C. Villemez. *AWS Outage Analysis: December 7, 2021.* https://www.thousandeyes.com/blog/aws-outage-analysis-dec-7-2021. 2021 (cit. on p. 121).

[294] J. Greene. *Amazon's cloud-computing unit problems take down websites, services.* https://www.washingtonpost.com/technology/2021/12/07/aws-outage-websites-offline/. Dec. 2021 (cit. on p. 121).

[295] I. Steger. *How Amazon Outage Left Smart Homes Not So Smart After All.* https://www.bloomberg.com/news/articles/2021-12-08/amazon-outage-sparks-anger-as-fridges-stop-people-locked-out. 2021 (cit. on p. 121).

[296] A. Akhtar. *Furious customers blast Amazon as an outage knocks Ring doorbells, baby monitors, and Alexa products offline.* https://www.businessinsider.com/ring-home-monitoring-services-down-aws-outage-2021-12. Dec. 2021 (cit. on p. 121).

[297] Cisco Crosswork Cloud. *BGPStream.* https://bgpstream.crosswork.cisco.com/. 2022 (cit. on p. 122).

[298] FireHOL Porject. *FireHOL IP Lists | IP Blacklists | IP Blocklists | IP Reputation.* https://iplists.firehol.org/. 2022 (cit. on p. 122).

[299] W. Wegner. *IoT Platform Companies Landscape 2021/2022: Market consolidation has started.* https://iot-analytics.com/iot-platform-companies-landscape/. 2021 (cit. on p. 123).

List of Figures

List of Tables