NORTHWESTERN UNIVERSITY

Revising System Premises For a Secure and Private Web

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Yunming Xiao

EVANSTON, ILLINOIS

June 2024

ABSTRACT

As residential bandwidth continues to grow and people become increasingly dependent on the Internet, previously overlooked security and privacy issues have become severe threats to Internet users, raising public concerns. These threats are numerous and far-reaching, spanning across various network or system layers, as well as different applications and services. Despite the considerable efforts made, the current situation is still far from ideal, mainly because of the continuously evolving security and privacy demands, as well as the rapid development of countermeasures.

This thesis aims to examine the security and privacy vulnerabilities in the current Web components and present appropriate solutions to address them. My approach is to develop practical cutting-edge network systems that enhance security and privacy without compromising on efficiency. To achieve this objective, I undertake comprehensive evaluations to gain insights into the current systems, and suggest enhancements through revising the key system premises that contribute to the vulnerabilities. Such revisions facilitate the mitigation of vulnerabilities and allow the incorporation of new system pieces, *e.g.,* advanced cryptographic tools. The resulting system should also provide satisfactory performance and be feasible to be deployed today.

In this thesis, I begin by examining one general concept for securing any network system - the virtual private network (VPN). Specifically, I concentrate on decentralized VPN (DVPN), a recent system revision proposal that improves user privacy by distributing VPN proxies to multiple parties, making it challenging for anyone to retrieve information about users and connections.

Nonetheless, my analysis reveals that DVPN fails to provide adequate security and privacy guarantees due to the reliance on trust placed in the proxies.

Unfortunately, relieving such trust dependencies in a proxy-based system proves to be a formidable task. Hence, I pivot to directly enhancing the specific Internet services and applications themselves. My first target is the domain name service (DNS), a fundamental Internet service behind almost any wild-area network requests. Through careful scrutinization of existing proposals, I find that the presence of recursive resolvers is the key system premise that contribute to the current DNS privacy vulnerabilities. DNS privacy can be substantially improved by having the recursive resolvers operate *in blind*, *i.e.,* answering the queries without knowing their contents. While this sounds counter-intuitive, it can actually be realized through advanced cryptography tools named private information retrieval (PIR). Following this direction, I build PDNS, an efficient and practical DNS system based on PIR. Thorough evaluations demonstrate that PDNS achieves acceptable performance as of today and offers better privacy preservation than any state-of-the-art proposals.

PDNS cannot safeguard the interconnected and multi-layered Internet alone. Indeed, user privacy exposure at one component negates protective measures at other components. With this in mind, I proceed to investigate another vital Internet component, the Hypertext Transfer Protocol (HTTP) – the bedrock of the Web – where I identify an instance of such a privacy violation. Specifically, Web providers are harvesting user information through HTTP cookies. The root cause is that current cookies are semantically oblivious, *i.e.,* they carry personally identifiable information (PII) but lack useful information for analytics. To solve this issue, I revise the system premises and propose semantic cookies – cookies discarding any PII. Evaluations show that semantic cookies not only patch up privacy leakage in HTTP requests but also accelerate cookie analytics by exploiting edge infrastructures, providing incentives for adoption. This exemplifies my system design philosophy of achieving balance among performance, security/privacy guarantees, and practicality.

THESIS COMMITTEE

Aleksandar Kuzmanovic Northwestern University	Committee Chair
Fabián Bustamante Northwestern University	Committee Member
Yan Chen Northwestern University	Committee Member
Matteo Varvello Nokia Bell Labs	External Committee Member
Xiao Wang Northwestern University	Committee Member

ACKNOWLEDGEMENTS

Success is not a solitary achievement but rather the culmination of collective support and collaboration. Among all, I wish to express my sincerest appreciation to my advisor, Aleksandar Kuzmanovic, for his profound expertise and supportive mentorship throughout my entire graduate school journey. I have come a long way from learning the ropes of research to striving towards becoming an independent researcher. The path has been both enlightening and enjoyable, largely due to his willingness to grant me significant freedom in exploring my intellectual curiosities, while guiding me to make prudent decisions at critical junctures. In retrospect, I could not have envisioned a more fulfilling Ph.D. experience.

I am also immensely grateful to Matteo Varvello, whose meticulous attention to detail, hands-on coding expertise, and active participation in research discussions significantly contributed to my Ph.D. journey. In many ways, he has been akin to a second advisor to me. Most of my Ph.D. projects, notably some of the most impactful ones, owe much to his involvement, as does my enriching first internship at Nokia Bell Labs.

I wish to thank Congcong Miao, for our enduring collaboration that has opened up new research directions, distinct from my academic work at school. This gratitude also extends to Ruiqiang Dai, Shengli Zheng, Yadong Liu, Dongbo Gu, Miantao Wan, Yinchao Yang, and all other colleagues at Tencent for fostering a collaborative and productive environment. Additionally, I would also like to thank Marco Canini and Zaoxing Liu for their assistance in multiple projects and their help

during my job search.

I also wish to thank Chenkai Weng, not only for our friendship but also for our fruitful collaboration on the PDNS project, which would not have reached its successful conclusion without his support. Our discussions have broadened my academic horizons, steering me toward the fields of security and privacy, which have become one of the major focal points of my research.

I extend my heartfelt gratitude to the rest of my thesis committee members, Fabián Bustamante, Yan Chen, and Xiao Wang, for their invaluable insights and guidance at various stages of my doctoral journey, each contributing uniquely to my academic development.

My time at HPE, alongside Puneet Sharma, Diman Zad Tootaghat, Aditya Dhakal, Lianjie Cao, and others, has been immensely educational, exposing me to new research directions and working styles. I am equally thankful to the families who supported me during my summer in Milpitas, California, making it a memorable part of my Ph.D. experience. Furthermore, the mentorship by Qiushi Wang, Anthony Sofia, and other colleagues at Google provided a remarkable life experience in New York City and an industry perspective that, while I remain inclined towards academia, has enriched my professional outlook.

I would also thank the students I had the honor of mentoring. These relationships have not only contributed to our collective research endeavors but have also significantly enhanced my mentoring skills. I also extend my gratitude to Pengfei Wang, Sen Lin, and Yanzhi Li for our engaging discussions and collaborative efforts.

To my friends, your companionship throughout this journey, especially during the challenging times of the COVID pandemic and beyond, has been a source of immense comfort and strength.

Finally, my deepest appreciation goes to my parents, whose unwavering support and belief in my aspirations have been the bedrock of my achievements.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

There has been growing public attention and concerns about security and privacy issues when using Internet applications and services. Indeed, the Internet was born several decades ago for the sole purpose of connection. Back then, the primary research and engineering efforts were focused on improving the performance of the connections. Few people would rely on it for their day-to-day activities. Yet, the Internet has evolved quickly over time and it has outlined a completely different landscape today: the residential bandwidth has increased by thousands of times – from a few Kbps to around 100 Mbps in most developed countries today [33] – and the Internet services have become an integral part of billions of people's daily routines [151], from communication and entertainment to work and commerce.

As the reliance on the Internet continues to grow, the once-neglected security and privacy issues have become serious threats to Internet users. These threats are diverse and widespread, extending across various network layers and different applications/services: from pervasive monitoring of plaintext traffic [168] to network middleboxes attempting to decrypt encrypted channels [58, 114, 132, 133, 134, 207, 223, 282, 289, 291, 310], from information exposure of domain name service (DNS) [45, 178, 263, 285, 314] that acts as the "Internet phonebook", to privacy leakage caused by HTTP headers/cookies [213, 313, 315] that originally aimed at customizing Web experiences, from vulnerabilities in mobile devices [222, 302, 326] to exploitations of Internet of Things (IoT) devices [97, 104, 197, 198, 201, 276, 303, 311, 313], and more.

Great efforts have been made to resolve these issues. For example, the virtual private network (VPN) [281] and its variants [40, 160, 250] are proposed to hide user identity. Next, the end-to-

end encryption scheme TLS [158]/HTTPS [265] has been widely adopted by Web browsing [12] and gradually supported by DNS [187]. Further, more advanced cryptography techniques such as differential privacy [166] and multi-party computation [145] have been applied to collect user logs while preserving user privacy.

Despite these efforts, the security and privacy guarantees provided by current network systems are not ideal. On the one hand, the demand for security and privacy is continuously rising as users' needs evolve, and countermeasures continue to develop. As a result, the deployed security and privacy mechanisms periodically fall behind the state of the art and require updating. On the other hand, some of the state-of-the-art cryptographic tools face obstacles in integration with current network systems, hindered by performance constraints and contradictions with established system premises, thereby impeding their gradual adoption.

1.1 Thesis Statement

This thesis aims to lay the groundwork for the next generation of Web systems by critically examining the current security and privacy vulnerabilities of existing systems, identifying key system premises that relate to these vulnerabilities, investigating whether such system premises are in contradiction with the latest cryptographic tools, exploring the system designs that alter the key system premises, and evaluating the trade-offs among performance, security and privacy improvements, and practicality of the new system designs.

Thesis Statement

Devising the best next Web systems with enhanced security and privacy hinges on judicious revisions of system premises, which alter key system behaviors to eliminate security and privacy vulnerabilities while ensuring satisfactory performance and practicality.

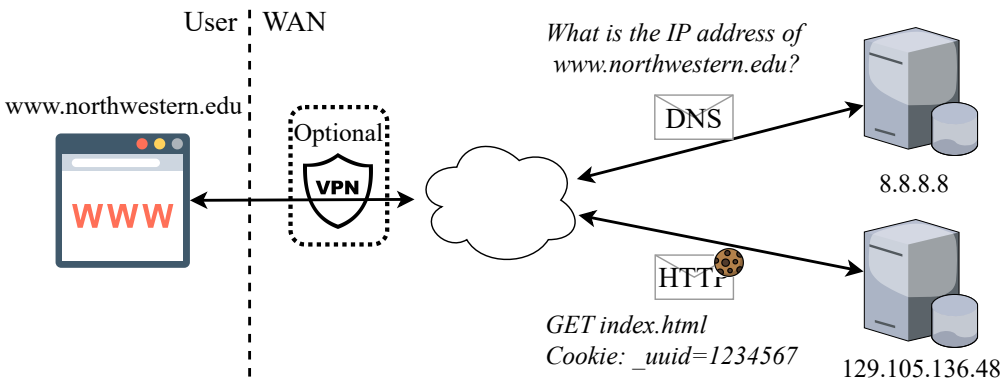


Figure 1.1: Visualization of the basic path of a Web request, representing the scope of this thesis.

1.2 Thesis Contributions

Notably, given the interconnected and interdependent nature of different network layers and applications, the security and privacy guarantees of network systems follow the Cannikin Law that every component must be resilient for the guarantee to hold. For instance, sending a request with a Web cookie that contains personally identifiable information to a Web server can expose the user's identity, regardless of what other measures at different layers, such as VPN at the network layer, are taken. Therefore, this thesis explores multiple network systems across different layers and applications.

Specifically, I concentrate on three critical and widely-used network systems that function as components of the Internet along the path of a Web request: VPN, DNS, and Web cookie analytics, as illustrated in Figure 1.1. In the first project, I conduct a comprehensive evaluation of the ecosystem of the decentralized VPN – an already realized system revision to centralized VPN systems – through extensive measurements. Subsequently, I propose novel system revisions for DNS and Web cookie analytics, aiming to enhance security and privacy assurances while maintaining optimal performance and practicality, in alignment with the thesis statement. It is noteworthy that

this thesis serves as a start rather than the conclusion of a continuous effort to strengthen network security and privacy.

Measurement of both the existing network system and potential enhancement toolkits is an indispensable part of the exploration process, serving as the first step in my approach. The next critical step is to locate the key system premises that contribute to the vulnerabilities in current network systems based on careful analysis of the measurement results. Then, I thoroughly explore hypothetical scenarios involving alterations to these key system premises, answering "what-if" questions. Proceeding with further evaluation to ascertain the trade-offs elucidated in the thesis statement, I then determine the most appropriate option from the system design space and proceed to build and, if possible, deploy the new system.

1.3 A First Look Into Decentralized VPNs

VPN [281] is a widely used method for preserving user anonymity and safeguarding privacy. Today, most VPN providers host multiple servers as proxies and serve numerous users. However, there are no mechanisms in place to prevent the proxies from learning the destinations of each user, potentially giving rise to privacy concerns unless the users unconditionally trust the VPN providers, which is an unrealistic assumption. Additionally, since VPN providers have a centralized structure, it is easy to identify their nodes, which undermines the effectiveness of VPNs in bypassing censorship [206].

Fortunately, there have been many proposals to solve this. In this thesis, I focus on the decentralized VPN (DVPN), which is an upgraded version of the prevalent centralized VPN and has recently gained much popularity. DVPN introduces a revision to the system premises that traditional centralized VPNs hold, where the VPN nodes are provided and managed by one party. This revision provides enhancements to security and privacy guarantees by scattering traffic across

many parties, making it harder to recentralize the information if needed.

Several systems [17, 21, 24, 25, 32, 35] have realized this idea, and they have attracted increasing user bases by exploiting the trend of increasing idle traffic of Internet users [34, 39, 116, 268, 275], enabling them to monetize their spare bandwidth. Despite the proliferation of such systems, little is known about their performance, security and privacy guarantees, and how such marketplaces operate. For instance, it is unclear what the key factors are that determine the price of spare bandwidth and how such prices differ worldwide. I thus investigate the ecosystem of DVPNs shedding light on the above questions [312]. My observations motivated me to create RING, a first and concrete system that helps to enhance the current DVPNs.

Despite the efforts, I find that DVPN still cannot provide satisfactory security and privacy guarantees to its users given that it cannot prevent the DVPN nodes from monitoring the bypassing traffic. This problem is intrinsically hard to solve without full control of the entire network. An alleviation could be to employ multiple proxies before reaching the destination, preventing each proxy from learning the identity of the user and destination at the same time. This can be accomplished through complicated user setups [22, 105] or by using Tor [40, 160], which directs user traffic through multiple proxies in a volunteer overlay network. However, this enhancement incurs performance penalties [93, 228, 251] due to encryption/decryption and the need to traverse multiple proxies before reaching the destination. Tor or similar setups are also not free from security issues or privacy leaks. For example, to preserve privacy guarantees, Tor requires that a large portion, *e.g.,* $>75\%$ [299], of the bandwidth capacity is controlled by honest proxies.

In conclusion, VPN, which involves introducing a proxy between two end hosts, is a fundamental concept for enhancing security and privacy in network systems and can be applied almost universally. Several variations and improvements have been proposed, such as DVPN – which I have extensively explored in this thesis –, Tor, and more. However, all of these solutions require

some trust assumptions about the proxies, or their security and privacy guarantees will fail, leaving them inadequate for the growing security and privacy concerns and requirements. Therefore, proxies can no longer solely be relied upon. I instead re-focus my attention on Internet applications and services themselves, searching for solutions that offer more robust security and privacy enhancements that meet current expectations.

1.4 Enhancing DNS Privacy with Private Information Retrieval

I shift my attention to the Internet applications/services themselves. My first target is the DNS for two reasons. First, it represents one of the Internet's cornerstone services. Second, the latest advancements in DNS privacy enhancements hinge on proxy trust, echoing the challenges inherent in VPN scenarios.

To provide a clearer explanation, let us begin by understanding what DNS is. Before any application requests are sent, they need to locate the destinations. This process relies on a key infrastructure of the Internet – the DNS [49], which translates human-readable domain names to machine-understandable IP addresses. DNS privacy has been neglected for years. Recently, DNS over HTTPS (DoH) [187] has improved the situation by fixing the issue of in-path middleboxes. Further progress has been made with proxy-based solutions such as Oblivious DoH (ODoH) [285] and DoH over Tor (DoHoT) [245, 246], which separate a user's identity from their DNS queries. Nevertheless, these solutions rely on trusted DNS resolvers and non-collusion with the proxy network.

In general, the issue of privacy leaks primarily occurs at the DNS resolvers. It follows that the solutions involve altering this key system premises – by either removing the ReRs from DNS [278], or having ReRs operating *in the blind*, *i.e.,* by resolving domains without knowing what they are. The former option exhibits high performance penalties to users, amplifies workloads on the

ANSes, and raises additional security concerns. The latter option seems counter-intuitive, but in reality several techniques exist which allow similar operations. These techniques fall in the branch of Private Information Retrieval (PIR), which is achieved by various cryptographic tools such as homomorphic encryption [113, 172, 173, 264, 264].

After assessing both strategies, I concluded that eliminating RRs from DNS is impractical due to its prohibitive performance impacts and security vulnerabilities. Conversely, incorporating PIR techniques into DNS is a viable option. Driven by these findings, I introduce PDNS [314], which, to the best of my knowledge, is the first practical initiative enabling ReRs to function privately through the use of single-server PIR technology [241]. PDNS aims to *augment* rather than substitute DNS, akin to DoH and ODoH, but it offers superior privacy protection, including safeguards against collusion and analysis of regional access patterns.

1.5 Semantic Cookies for Anonymous Online Streaming Analytics

Still, security and privacy guarantees cannot be realized if any other Internet component fails to provide them. The HTTP request – another bedrock of Web – is one such component where I have identified a flaw in the current practice: the current HTTP cookies are *semantic-oblivious*, *i.e.*, carrying personally identifiable information such as user IDs [135, 252]. If the user sends its user ID to others during HTTP sessions, its privacy will be exposed despite the efforts of adopting PDNS or using VPN. The user IDs have allowed the service providers to record any information about the individual users as much as they can for an indefinite duration as long as the users do not actively clean up – and most users are not aware of it at all [273]. It follows that when the user sends the HTTP requests with semantic-oblivious cookies, privacy preservation will become incomplete.

The solution to this problem resides in the trend of migrating infrastructure towards the edge.

More concretely, in recent years, we have witnessed a growing trend of content hyper-giants deploying server infrastructure and services close to end-users [190, 214], in "eyeball" networks [175]. This results in a separation between where the contents are served and where the user cookies are analyzed, *i.e.*, the users will first access the edge servers, *e.g.* CDN servers, to obtain the Web-page contents whereas the user cookies will be forwarded to a remote data center before being processed. The inability to process the user cookies at the network edge is caused by a common setting where user profiles, necessary for analytics, are stored deep in the data center backends. This setting also carries privacy concerns as the user profiles – which are personally identifiable – are communicated to the Web providers, yet the users are almost blind to what data is associated with their identities and how the data is analyzed.

In this thesis, I propose to revise the system premises by breaking this arrangement and planting encrypted *semantic cookies* at the user end [315]. Without altering any of the existing protocols, this enables capturing and analytically pre-processing user cookies soon after they are generated, at edge ISPs or content providers' off-nets. More importantly, it ensures that user anonymity is preserved during the analytics. I have demonstrated that it is viable to encode semantic cookies in the existing application or transport protocols.

I further present Snatch, a system that coordinates all the components involved in online streaming analytics with semantic cookies. It helps to configure the early forwarding and pre-processing procedures at the network edge to speed up the online streaming analytics and preserve user anonymity. Evaluations of Snatch – based on real-world measurements – show that when processing can be done early in-network, Snatch can speed up user analytics by 10-30x. Given the growing trend of migrating infrastructure towards the edge, such speedups along with privacy enhancements are likely to soon become a reality.

To summarize, combining the semantic cookies and PDNS introduced earlier, this thesis builds

a key step towards a more secure and private Internet.

1.6 Thesis Organization

The remainder of this thesis is arranged as follows. In Chapter 2, I detail the exploration of the decentralized VPN ecosystem. Then, in Chapter 3, I present my design of PDNS system, which introduces practical security and privacy enhancements to the key infrastructure of the Internet – the DNS system. Next, I address the issue of privacy leaks from conventional HTTP cookies, proposing a novel architecture designed to improve both performance and privacy in online streaming analytics in Chapter 4. Finally, the thesis concludes with Chapter 5, where I summarize the key contributions of my work and offer concluding thoughts.

CHAPTER 2
A FIRST LOOK INTO DECENTRALIZED VPNS

VPNs offer two major services: (i) encrypted traffic between the user (VPN *user*) and a VPN *node*, and (ii) a new public IP address (the one from the selected VPN node). Users leverage these two services in different ways. Encryption is beneficial, for example, when connecting to a network that the user does not trust, *e.g.,* a public WiFi or a potentially sketchy ISP. Obtaining a new IP address is mostly useful when the user aims at bypassing censorship or geo-location blocks.

Since privacy is a major concern of VPN users, there is one potential flaw with today's centralized VPNs. The user needs to inherently trust the VPN provider not to interfere or log any of their personal traffic. It is to be noted that VPN providers are commercial entities that might offer their services relying on other commercial entities, *e.g.,* they could use multiple cloud services to obtain a worldwide footprint. It follows that even trusted and respectable vendors might unknowingly incur in issues with a specific provider ranging from surveillance, misconfiguration, and even hacking. Either of these issues can compromise the user privacy.

In [206] the authors actively investigate 62 commercial VPN providers and find unclear policies for no logging, some evidence of tampering with their customer traffic, and a mismatch between advertised VPN node locations and actual network location. In many cases, this misbehavior was not purposely performed by the VPN provider but was caused by some misconfigurations. When contacted by the authors, all providers quickly reacted to fix the reported misconfigurations.

The above issues are key motivations behind a new trend: decentralized Virtual Private Networks (DVPN). In a DVPN nodes can be both client and node, in the sense that users have the option to offer a portion of their upload bandwidth to carry traffic on behalf of DVPN users. For

example, assuming Alice (France) wants to access some content only available in the US, she can piggyback on Bob's residential IP address (US). A client would discover available DVPN nodes either via a central repository or by using a distributed repository [307].

Since then, a few DVPN networks, such as Hola [11] and VPN Gate [43], have been developed and gained popularity among users. Hola users either pay a monthly premium or contribute a portion of their upload bandwidth to other Hola users. VPN Gate, on the other hand, relies solely on volunteer machines. However, depending on temporary users or volunteer machines cannot guarantee a reliable service. Furthermore, acting as a proxy and offering services to other users may result in legal consequences for volunteers, which can lower people's interest in DVPNs.

Fortunately, two trends are currently aiding the growth of DVPNs while also reshaping them: (i) the increasing availability of spare bandwidth and (ii) the emergence of blockchain and cryptocurrency [304]. According to Ookla [34], average Internet speeds in American homes grew 20x in the last 10 years, from 8 Mbps (2010) to 180 Mbps (2021). A similar trend is observable worldwide [33]. Several recent studies [39, 116, 268, 275] suggest that the added cost for faster Internet speeds — *e.g.,* \$50 monthly to boost from 200 to 300 Mbps with Comcast Xfinity [14] — is not worth it to most residential users, as only a median of 5% of bandwidth is used. For example, a family of 4 concurrently streaming HD videos only requires $\sim$20 Mbps, not to mention unused bandwidth at night.

Meanwhile, cryptocurrencies on top of blockchain technology provide a convenient payment method to individuals worldwide. The combination of these two trends has led to the emergence of several DVPN networks that offer mechanisms for users to monetize their unused bandwidth [17, 21, 24, 25, 32, 35]. Effectively, these DVPNs are building *bandwidth marketplaces* where spare bandwidth is auctioned. For example, Alice (France) is willing to pay Bob (US) \$1 to tunnel her video traffic and avoid geo-blocking.

While such systems are attracting a considerable number of users, both as clients (buyers) and providers (sellers) of spare bandwidth, little to nothing is known about security and privacy guarantees of the DVPNs, the properties of such marketplaces, and the dominant factors that affect them. Understanding the properties of existing marketplaces helps to shed light on the future market evolution as well as designing more reliable and efficient ones, which is a topic I explore in this chapter. Further, existing marketplaces can help us understand which value buyers and sellers associate with (spare) Internet bandwidth today.

The following lists my major contributions:

- I conduct the first comprehensive investigation on the DVPN ecosystem, revealing their security and privacy characteristics as well as characterizing their footprint, performance, and pricing schemes.
- I formalize the bandwidth monetization problem by considering a single-vendor bandwidth marketplace, and analyze the price ranges that create the most efficient marketplace.
- I extend the modeling to a multi-vendor bandwidth marketplace. I further realize such a marketplace with RING, a software that offers its users fine-grained control on security and how and where to monetize their spare bandwidth.

2.1 Background

Decentralized VPNs (DVPNs) — a new form of VPN with no central authority — are enhancements to centralized VPNs. DVPNs scatter the user traffic across many parties and hence increase the difficulty to recentralize the information if needed. DVPNs are also realizations of single-vendor bandwidth marketplaces. That is because, in essence, a DVPN is a tool that allows users to sell their (spare) bandwidth.

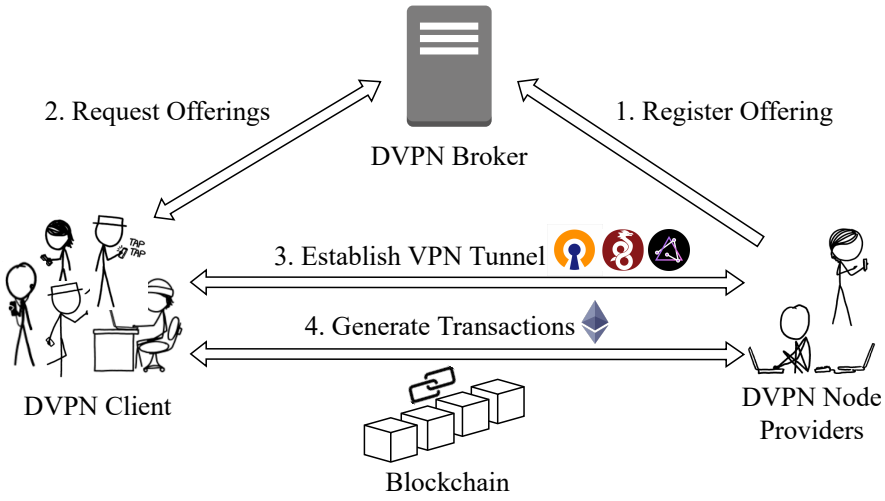


Figure 2.1: Visualization of functioning of DVPN that adopts cryptocurrency as the payment method.

DVPN users can have two roles, either concurrently or disjointly: *node* and *client*. A user acts as a *node* when it forwards traffic on behalf of other users and requests some form of compensation for this. A user acts as a *client* when it pays for its traffic to be tunneled via a DVPN node of her choice.

Over the years, there have been many attempts at building DVPNs. For instance, Hola [11] is the first DVPN to the best of my knowledge. Hola requires the users to pay a monthly premium or to contribute a portion of their upload bandwidth to other Hola users before being able to use its services. VPN Gate [43] is a DVPN originated as a research project [250] to achieve blocking resistance to censorship firewalls. However, it relies solely on volunteer machines. However, I do not include Hola or VPN Gate in the exploration because they cannot guarantee reliable services depending on temporary users or volunteers. They are neither a good approximation of the actual bandwidth marketplace given their users cannot charge for their bandwidth.

I instead focus on recent DVPNs which originated in conjunction with the rise of blockchain and cryptocurrency [144, 304]. In particular, DVPN nodes can directly determine their prices

Table 2.1: Summary of current DVPN solutions.

DVPN	Client Platform	Node Platform	Open Source	Payment Scheme	Tunneling Protocol	Comments
Mysterium	Android, Mac, Windows	ARM, x86	Yes	Data transferred, connection time	OpenVPN, Wire-	–
Sentinel	Android, Mac, Windows, Linux	ARM, x86	Yes	Data transferred	Guard WireGuard	–
Tachyon	Android, Mac, iOS	x86	No	Data transferred, staking reward	Tachyon	Still in development
Privatix	Android, Mac, Windows, Linux	ARM, x86	Yes	Data transferred	OpenVPN	Cannot join [59]
Orchid	Android, Mac, iOS	ARM, x86	Yes	Not yet released	OpenVPN, Wire-	Still in development
Lethean	X	ARM, x86	Yes	Data transferred	Guard OpenVPN	Dead project

and users are granted visibility on information like price charged, *e.g.,* cryptocurrency per GB, node location, and expected bandwidth (Mbps). Examples of such DVPNs are: Mysterium [21], Sentinel [32], Privatix [25], Tachyon [35], Orchid [24], and Lethean [17].

Figure 2.1 depicts the working procedure of such dVPNs. First, DVPN nodes register their "offering" (*e.g.,* location and cost per GB) at the DVPN broker. When a client wants to use the DVPN service, she requests the currently available offerings from the DVPN broker and selects a node to establish a VPN tunnel to. The tunnel is established directly between the client and the node. In the meantime, transactions are generated per the agreement between client and node, and are executed and recorded on the blockchain. It thus follows that recent DVPNs fit in the category of decentralized applications (dapps) [96].

Table 2.1 provides an overview of the above DVPNs. The table shows that Android is the most common client platform. With respect to the node, all DVPNs except Tachyon are open source and offer executables for both ARM and x86. The payment scheme of most DVPNs is based on how much data is transferred through the node. Mysterium also charges clients by how long they connect to a node. In addition, Tachyon pays a *staking* reward; "staking" refers to the

fact that the seller needs to put down some amount of cryptos (a stake) before they can receive rewards. In terms of VPN tunneling, OpenVPN [23] and WireGuard [47] are the most popular protocols adopted. The only exception is Tachyon, which uses a proprietary protocol also named "Tachyon" [16].

In green, I have highlighted the DVPNs which I have selected for both my active and passive monitoring: Mysterium, Sentinel, and Tachyon. These DVPNs were selected since they offer stable client and node implementations, and a payment scheme which is representative of a bandwidth marketplace. Privatix (orange) was instead only studied *actively*, *i.e.,* by leveraging its client to test its current nodes, since my (multiple) attempts to join the Privatix network as nodes have not been successful. Finally, Orchid and Lethean (red) could not be studied for the following reasons. Orchid has not opened node registration to regular users but only to partners; further their client was quite unreliable during automation. Lethean has no working client and its *staking* account, from which users need to acquire funds, is currently unavailable.

2.2 Methodology

I here describe my methodology to explore the DVPNs selected above and collect the data needed to populate my model of a generic single-vendor bandwidth marketplace. My rationale is to both *actively* and *passively* collect data from a DVPN. Active experiments consist of automating a DVPN client to send traffic via the available nodes. This is useful to learn about their footprint, pricing, and performance. Passive experiments consist of contributing bandwidth to such DVPNs by running several nodes. I are interested in characterizing how much traffic a typical DVPN node carries, and thus how much revenue (cryptocurrency) it could generate. Further, I want to explore the traffic *characteristics*, *e.g.,* presence or lack of harmful traffic, to help asses the *risk* associated with running a DVPN node.

2.2.1 Active Measurement

As per Table 2.1, my active experiments rely on Android DVPN clients because Android is the common platform among all dVPNs I aim to investigate. I have further confirmed that there are no significant differences between the information (node locations and prices) available through the different client platforms. My testbed consists of an Android device (a Samsung S9 running Android 10) controlled by a Raspberry Pi 4. The Android device is used to run the DVPN apps, while the Raspberry Pi realizes the automation, *e.g.,* launch a DVPN app and select a node to connect to. I chose the Raspberry Pi for its convenience and given that its task is simple and more powerful hardware is not required. The Android device connects to a fast WiFi (80 Mbps upload/download bandwidth) and is located in North America.

I automate DVPN usage via the Android Debugging Bridge (ADB [52]), a rich Android protocol which allows to automate app operations like launching, scrolling, and GUI interaction. At a high level, I use ADB to instrument each DVPN app to automatically iterate through its available nodes, while attempting a connection. I rely on visual inspection of screen recordings to verify and learn how to iterate through all states each DVPN app can reach, *e.g.,* connection ready or more random states like *rate the app*, which I then translate into automation scripts.

I use several techniques to both gather information about a DVPN and enforce correct crawling functionalities. For example, I monitor Android network interfaces to verify successful "connect" and "disconnect" operations. I also rely on Android logging (`logcat`) where developers often log information like state changes, node IPs, payments, etc. I further use screenshots, both XML of the information on screen (via `uiautomator`) and actual images coupled with OCR processing, to collect statistics which are only available on screen. When available, I also resort to public APIs, as in the case of Sentinel [76] which curiously even reports the CPU consumption measured at its DVPN nodes.

I use the above automation to build two active measurement campaigns: *discovery*, and *speedtest*.

Discovery – The goal of this measurement is to discover nodes offered by a DVPN, along with any public information like pricing and advertised bandwidth. This implies quickly iterating through the GUI of each DVPN (or query its public API, if available) logging information about node counts and locations. Since this method does not require to connect to each node, it is quite lightweight, and I thus run it daily over 6 months, from December 2020 to May 2021.

Speedtest – The goal of this measurement is to benchmark the *connectivity* (availability, location, and download/upload bandwidths) of the nodes offered by a DVPN. This requires connecting to each node discovered using the above procedure, to then perform a speedtest. Compared to the discovery measurement, this test is more complex and invasive. I thus resorted to run it monthly; further, in presence of very large DVPNs I sample a subset of the nodes by selecting a maximum of 10 nodes per country.

To perform speedtests, I leverage the public service offered by Netflix at `https://fast.com` automated via ADB. First, I configure a target dVPN node to be tested. Then, I launch the Chrome browser and visit the speedtest website. Last, I use `uiautomator` – which dumps content on screen in XML format – to retrieve measured bandwidths, latencies, estimated location, and server used for testing. I also take a screenshot of the page to retrieve the above information via OCR in case of failure of `uiautomator` (which can happen in presence of dynamic content on screen). To avoid very long and expensive tests, I limit the duration of each test to 10 seconds, which implies a maximum upload/download of 100 MB under my (residential) connectivity.

2.2.2 Passive Measurement

In these measurements, I run nodes for the main DVPN providers while passively collecting their traffic using Tstat [42], a popular traffic sniffer which automatically analyzes TCP and UDP traffic.

Tstat uses the classic 5-tuples[1] to identify TCP *sessions* and UDP *flows*. TCP sessions are identified using TCP connection establishment process. UDP flows are harder to detect since there is no explicit notion of a session. Tstat defines a UDP flow as the set of packets with same 5-tuples with inter-arrival times smaller than 200 seconds. I further call DVPN *session* a collection of TCP/UDP flows between client and node, and *outgoing sessions* the TCP/UDP traffic between DVPN node and destination IPs.

Given HTTPS represents the majority of today's Internet traffic [12], I rely on DNS – when not encrypted – and SNI – not yet encrypted even with TLSv1.3 [29] — for coarse traffic characterization, *i.e.,* I identify accessed domains but not, for instance, specific webpages. Next, I adopt McAfee domain classification service [6] which achieves the highest coverage according to [296], *i.e.,* 94% over 4.4 million domains. McAfee provides two attributes per domain: *reputation* and *type*. "Reputation" is calculated dynamically by the TrustScore system [27] and maps to four ratings: minimal risk (<15), unverified (15-30), medium risk (30-50), and high risk (>50). "Type" depends on the content available at a given domain, *e.g., facebook.com* corresponds to social networking. I further use signature matching provided by IPP2P to identify P2P traffic.

2.2.3 Marketplace Formulation

Here, I formalize the bandwidth monetization problem in a single-vendor marketplace. This will help us to comprehensively analyze the existing marketplaces and quantify if, and how sub-optimal, they may be. I assume that a *seller* at location l offers her spare bandwidth to potentially multiple concurrent buyers. I further assume that the seller's Internet connection is characterized by some (spare) speed r, and a data cap D in a period of time T. For example, Comcast Xfinity offers download speed up to 1,200 Mbps, depending on the monthly price tag, for a maximum

[1] $< IP\_SRC, IP\_DST, PORT\_SRC, PORT\_DST, PROTO >$

of 1.2 TB per month [48]. Note that r is the minimum between download and upload speeds, or $r = min\{r_u, r_d\}$. This is because a seller is not an end-point but a "middle-point", which is required to utilize both her upload and download bandwidth. For example, when a buyer downloads a 1 MB file from the Internet, for the seller, this translates to 1 MB of download data which then needs to be uploaded to the buyer. Thus, the seller carries twice as much data as the buyer, and the actual speed depends on where the bottleneck is between a seller's download and upload bandwidth.

I consider a charging scheme defined by the pair (x, y), which represents the *amount of data consumed* and *duration*, *e.g.,* (Gigabytes, seconds). This is reasonable and representative of what I have observed in several bandwidth marketplaces (see Table 2.1). In addition, the seller may or may not be willing to carry some "dangerous" traffic, *e.g.,* contacting IP addresses labeled unsafe by services like the Safebrowsing list [56]. Generally speaking, I assume a seller defines a blocklist $A = \{dst1, dst2, \ldots\}$, which includes the set of destination IP addresses which should be blocked.

I call S the set of sellers participating in a marketplace. Each seller $s_j \in S$ posts her service in the marketplace defined by a location l_j, price settings x_j, y_j, rate limit r_j and blocklist A_j. Buyers can see seller details $s_j = (l_j, x_j, y_j, r_j, A_j)$ and decide to buy, hence connect, or not. In the following, I formalize the optimizations from both a buyer and a seller perspective.

Buyers' Perspective – Assume a buyer is looking for bandwidth with average speed b for a duration u. The buyer is further looking for bandwidth from sellers within a set of locations L, and her traffic is directed to a destination set DES. The chosen seller $s_j \in S$ must satisfy $r_j \geq b, DES \subseteq A_j, l_j \in L$. The price P the buyer needs to pay to seller s_j is:

$$P = x_j \cdot b \cdot u + y_j \cdot u \tag{2.1}$$

The buyer naturally would like to minimize her cost, given equal performance. Let $S(b, L, DES) \subseteq S$ be a sellers set which matches a buyer's constraints. To minimize the buyer's cost, a seller can be selected by optimizing the following objective function:

$$\min_{s_j \in S(b,L,DES)} P(S) = \min_{s_j \in S(b,L,DES)} (x_j \cdot b \cdot u + y_j \cdot u) \tag{2.2}$$

Intuitively, the process consists of filtering the sellers by given constraints (b, L, DES). Then, for a demanded bandwidth b, the buyer should select the seller among the left sellers minimizing $x_j \cdot b + y_j$.

Sellers' Perspective – I assume that connection decisions of each buyer are independent events, and that the number of buyers is large. Under this assumption, the arrival process of bandwidth buyers would follow the Poisson distribution, *i.e.,* $\mathbf{n} \sim Poisson(N)$, where **n** is the number of buyers and N is the mean number of buyers within time T. Intuitively, N is a function of (x, y, r, l, A) since it depends on the service price, quality, location, and seller's blocklist.

Let **B** and **U** be the random variables of bandwidth and duration of incoming traffic sessions, respectively. The expectation of income **I** of a seller within a period of time T can then be formalized as follows:

$$\mathbb{E}[I(x, y, r, l, A)] = x \cdot \mathbb{E}[\mathbf{n} \cdot \mathbf{B} \cdot \mathbf{U}] + y \cdot \mathbb{E}[\mathbf{n} \cdot \mathbf{U}] \tag{2.3}$$

The seller would naturally like to maximize her income. I optimize the seller's income by adjusting the unit prices x, y and bandwidth speed limit r, *i.e.,* I maximize the following objective function:

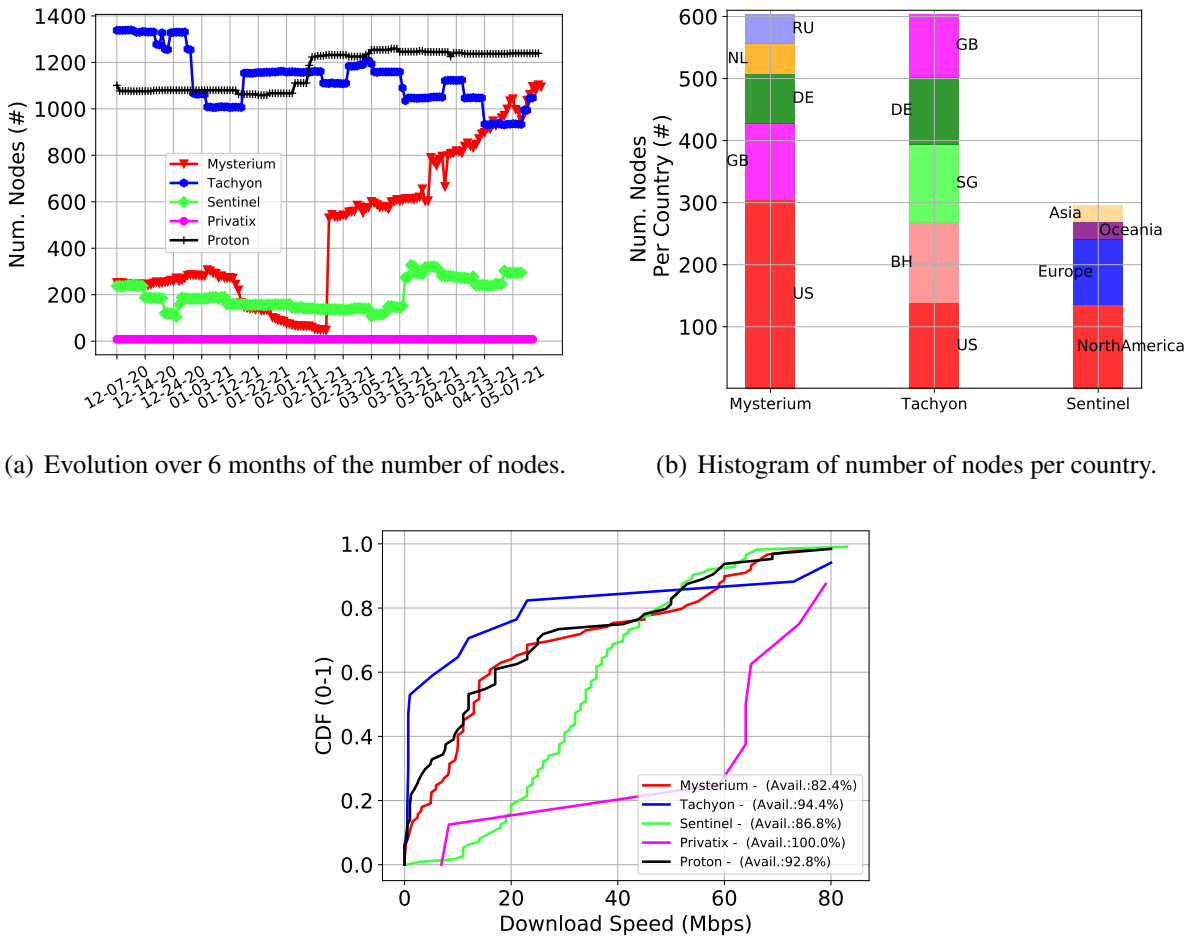
$$\max_{x,y,r} (x \cdot \mathbb{E}[\mathbf{n} \cdot \mathbf{B} \cdot \mathbf{U}] + y \cdot \mathbb{E}[\mathbf{n} \cdot \mathbf{U}]) \;\; s.t. \;\; \mathbb{E}[\mathbf{n} \cdot \mathbf{B} \cdot \mathbf{U}] \leq D/2 \tag{2.4}$$

2.3 Data Analysis

This section analyzes the data collected via my active and passive experiments. The analysis provides a detailed view of the DVPN ecosystem with respect to its footprint, performance, and traffic characteristics. I further investigate whether DVPNs are indeed concrete representations of a bandwidth marketplace, and the collected data-set can be used to model the variables contributing to the bandwidth monetization problem discussed in § 2.2.3.

Footprint and Performance – I start by investigating the *footprint* of the current DVPN ecosystem, *i.e.,* how many nodes compose each DVPN and where they are located. Figure 2.2(a) shows, for each DVPN, the evolution over the last six months (December 2020 – May 2021) of the *total* number of nodes advertised by each DVPN. The figure is further enhanced with data collected from ProtonVPN [26], a popular centralized VPN, given a basic account ($5 per month).

Figure 2.2(a) shows that, initially, only Tachyon had a footprint comparable with ProtonVPN, *i.e.,* in the order of one thousand nodes. However, Tachyon has lost 36% of its nodes over time while Mysterium's node count has been steadily increasing after February, and it is the largest DVPN with 1,100 nodes by the end of May. Mysterium has been *losing* nodes in January/February (reaching a minimum of 50 nodes), followed by a sudden increase to 530 nodes on 02-17-21. This behavior was an artifact due to Mysterium's migration from their version 1.0 to 2.0, which progressively made part of the network appears to be offline as discussed in [37]. Note also that ProtonVPN has added 100 news nodes in this time span. While Sentinel has also increased its footprint, it currently attracts a relatively small number of nodes compared to Mysterium and Tachyon. Finally, Privatix only counts 8 nodes, which are likely provided by the Privatix team given they are very stable and, in my experience, it is currently impossible to contribute a node to this DVPN.



(a) Evolution over 6 months of the number of nodes.

(b) Histogram of number of nodes per country.

(c) CDF of download speed with availability

Figure 2.2: Footprint and performance characterization of the DVPN ecosystem.

Next, I report on *where* DVPN nodes are located. Each stacked barplot in Figure 2.2(b) shows the top 5 countries per DVPN, as per 05-17-21. I omit Privatix whose eight nodes are located in: Toronto, Frankfurt, London, Bangalore, Amsterdam, Singapore, New York, and San Francisco. Note that Sentinel only distinguishes nodes by continent. The figure shows that, irrespective of the DVPN, the US (NorthAmerica for Sentinel) is the country where most nodes are located. Germany (DE) and Great Britain (GB) are two other popular countries among DVPNs. The geolocations of

DVPN nodes are reported by the DVPNs, and bias may exist [257]. In addition, 75% of the Mysterium nodes are *residential*, whereas the percentage of residential node drops to 45% for Sentinel and 0% for Privatix. I were instead unable to retrieve such statistic for Tachyon.

Finally, I report on the *performance* – in terms of download speed and availability – when using such DVPNs. Results for upload bandwidths are omitted since they exhibit a similar trend, although about half of the bandwidth available, overall. Figure 2.2(c) shows the Cumulative Distribution Function (CDF) of the download speed measured each month per DVPN (plus ProtonVPN); each VPN was tested independently at night, while making sure no local cross traffic was present. The figure shows that only Tachyon is overall slower than ProtonVPN. Mysterium has comparable performance with ProtonVPN while both Sentinel and Privatix significantly improve bandwidth, by up to 3x and 6x. The legend of Figure 2.2(c) also reports the overall availability of each DVPN which is, on average, comparable with ProtonVPN. High bandwidth and perfect availability offered by Privatix further confirm that its 8 nodes are likely managed by Privatix itself.

Traffic Characterization – Between February and April 2021, my nodes have served ∼505 thousand distinct buyers, ∼632 thousand DVPN sessions, ∼623 million TCP/UDP flows, accounting for about 16 TB of data. Download traffic is the highest contributor, about 10x the amount of upload traffic. Both residential and cloud DVPN nodes have attracted significant traffic over time (tens of thousands of sessions per DVPN), with the exception of the node located in Alibaba cloud (China) which has received no session via Sentinel, 604 sessions via Mysterium, and 1,924 sessions via Tachyon (see the bottom of Figure 2.3, CN-Seller). This is due to the interference of the great firewall [292].

Figure 2.3 visualizes from *where* DVPN traffic originates and is destined to, using Maxmind [20] to map `ip_src` and `ip_dst` at the country level. The middle of the plot shows the 10 machines – distributed between US, Italy, UK, and China – which were used for passive data

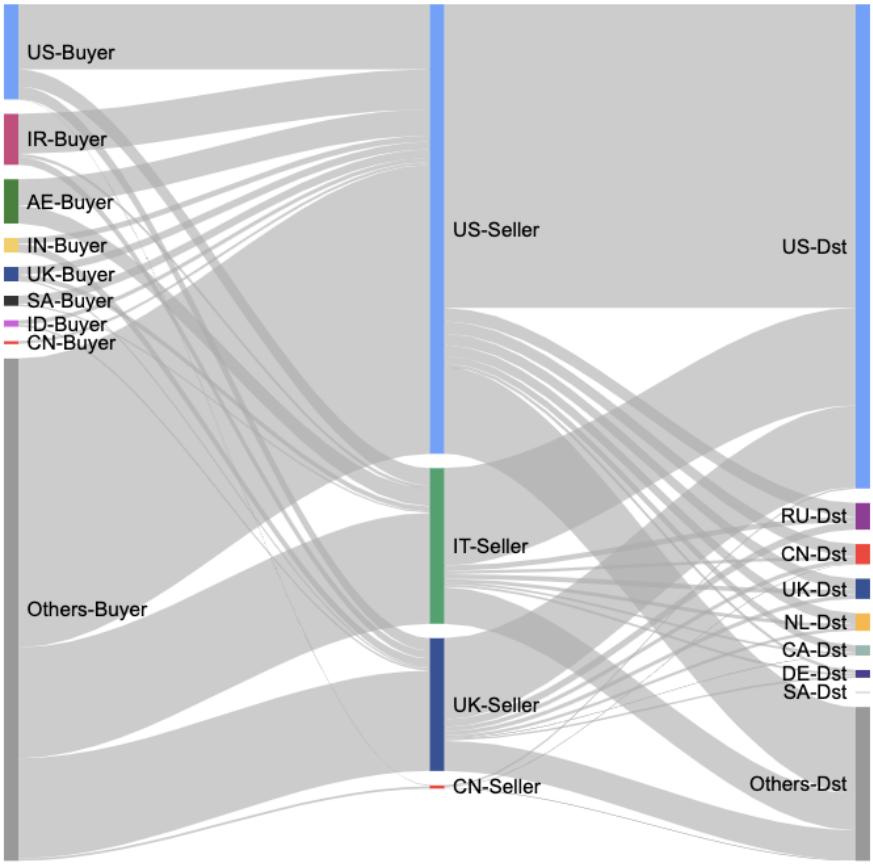
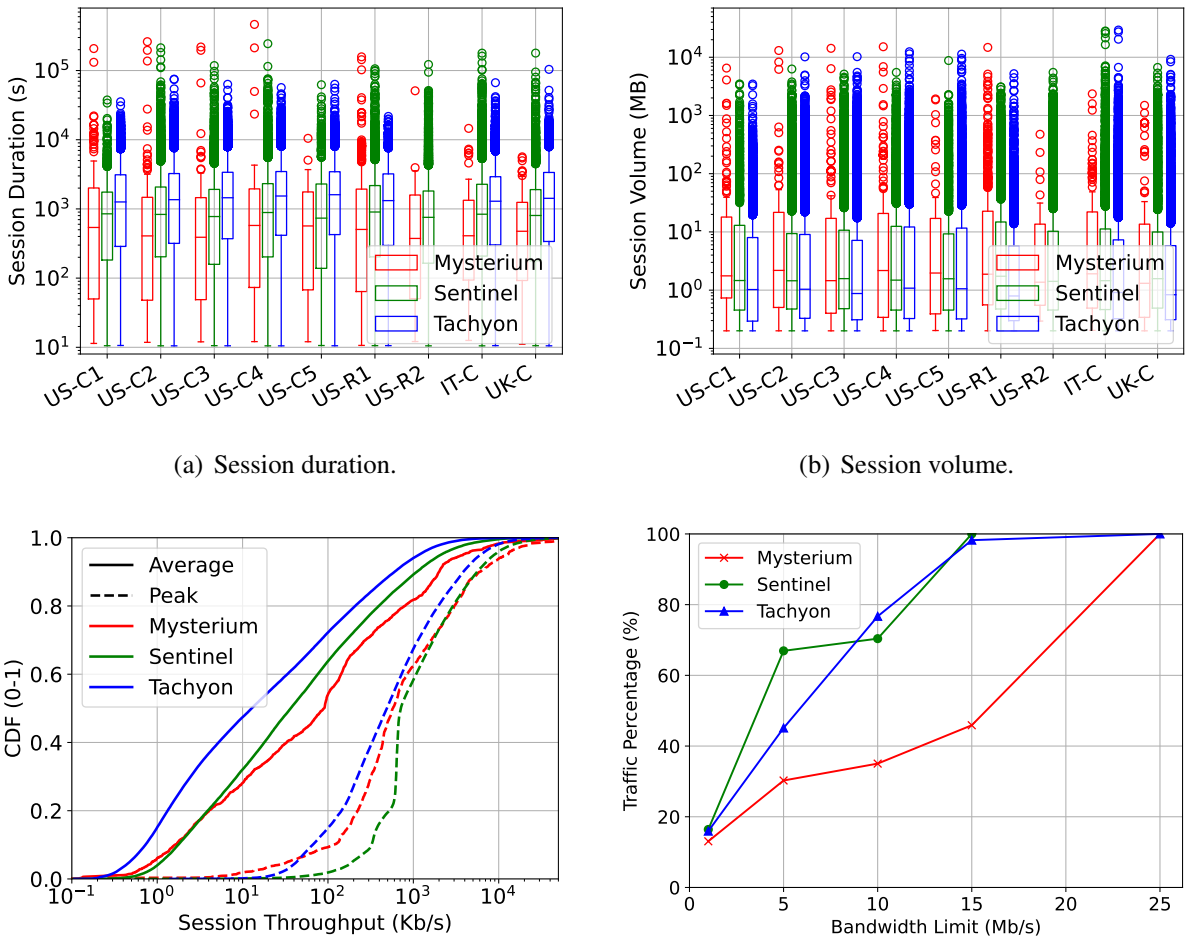


Figure 2.3: Visualization of DVPN traffic across Mysterium, Sentinel, and Tachyon. Buyer's locations are shown on the left, my machines where DVPN nodes are run in the center, and traffic destinations on the right.

collection. The figure aggregates data across the three DVPNs since no statistically meaningful difference was observed. The figure shows that the US has the most buyers, followed by Iran (IR), United Arab Emirates (AE), India (IN), and the UK, to complete the top 5 buyer locations. The US is also the most popular destination regardless of which node (middle of the plot) is used, accounting for over half of the traffic. Russia (RU) is the second most popular destination, followed by China (CN), UK, and Netherlands (NL).

It is noteworthy that the traffic destinations (right of Figure 2.3) may be biased because of the



(a) Session duration.

(b) Session volume.

(c) Session throughput.

(d) Traffic percentage.

Figure 2.4: Analysis of DVPN sessions: (a) duration, (b) volume, and (c) throughput. 9 node locations over 3 months; and (d) traffic percentage (ratio to the maximum observed) over one month as a function of a DVPN node available bandwidth.

presence of Content Delivery Networks (CDNs), which distribute the content to servers all over the world and allow users to retrieve the content from the closest location. That is to say, the destination of the same content may vary depending on the geolocation of the requesters, which are my nodes in this case. To quantify the impact of CDNs, I i) perform `whois` to check the registration of the destination IPs, and ii) validate whether the IP addresses of the destinations fall

in the range of CDN services when the company provides both cloud and CDN services, including Amazon [19], Google [9], and Microsoft [28]. I find that, depending on the node, traffic directed to CDNs ranges between 24 and 32%. Further, the most popular CDN providers, in descending order, are: Facebook, Akamai, and Cloudflare.

Next, I investigate *duration* and *volume* of the 629,156 DVPN sessions handled by my 9 nodes (see Figure 2.4[2]), omitting the Chinese node due to the lack of traffic discussed above. The figure shows two main results. First, user sessions are quite similar across nodes, with no significant difference apart from the tails (outliers in the boxplots). Second, user sessions are instead quite different among DVPNs, with Mysterium's sessions being overall shorter (median of 10 minutes versus 23 minutes for Tachyon) but carrying more traffic (median of 2 MB versus 1 MB for Tachyon). The latter result also implies that a large number of sessions (50% or more) are mostly idle. Nevertheless, many sessions carry a large amount of traffic, even up to multiple GB. Overall, the average sessions have a volume of 120, 60, and 40 MB for Mysterium, Sentinel, and Tachyon, respectively.

To further investigate the previous result, I derive the session *throughput* as the number of bytes transferred during a session divided by its duration. I further compute the throughput for each of the TCP/UDP flow within a session and select the maximum (*peak*) as an approximation of the bandwidth available to a user. Discounting my nodes upload bandwidth (since they are very well provisioned), the session throughput depends on two factors: i) user access bandwidth, ii) application demand, *e.g.,* if a user is reading an article online for a long time, very little traffic would be measured. Given that I did not notice significant difference among my nodes, Figure 2.4(c) shows CDFs of both session and peak throughput (per DVPN) among all my nodes. The figure shows a large discrepancy between session and peak throughput suggesting that application demand is the

[2]US-R2 is an ARM-based machine and thus does not support Tachyon, see Table 2.1.

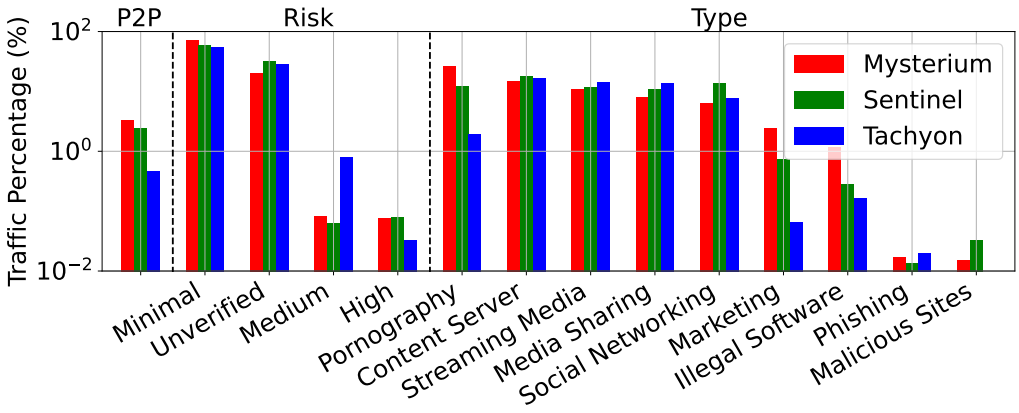


Figure 2.5: DVPN traffic composition according to McAfee and IPP2P classification.

main bottleneck, *i.e.,* users are mostly downloading from time to time rather than, for instance, streaming some video.

As a next step, I limit the upload bandwidth of my nodes and investigate the impact on the number of sessions. To do so, I set up, for one month, 5 additional AWS EC2 machines where I limit the bandwidth to 1, 5, 10, 15, and 25 Mbps for each DVPN. Note that the available bandwidth of DVPN nodes is not shown on the buyers' app. However, the buyers will likely switch to another node if the bandwidth of a connected node does not satisfy their needs. While it is natural to think that limiting the bandwidth would result in decreasing the number of sessions, Figure 2.4(d) shows that buyers from different DVPNs react quite differently. Mysterium buyers are the most sensitive to the bandwidth limits. For example, the number of Mysterium sessions is more than halved when implementing a 15 Mbps limit, which has no impact on both Sentinel and Tachyon buyers. This behavior is likely driven by a more demanding user-base, with overall higher bandwidth requirements (Figure 2.4(c)). Given all my nodes are equipped with more than 25 Mbps, this result further corroborates the above assumption that my nodes are not the reason of the trends observed above with respect to session characteristics.

Last, I characterize DVPN traffic composition according to classification based on McAfee and IPP2P (see § 2.2). Figure 2.5 shows, for each DVPN, the amount of traffic belonging to each category; the dashed vertical lines group traffic in higher level categories (P2P, reputation, and type). Regardless of the DVPN, P2P traffic is extremely low, accounting for less than 3% of the overall traffic. With respect to the traffic *reputation*, the figure shows that the majority of the traffic carries very low risk: 60-70% minimal risk and 20%-30% unverified, or in between minimal and medium risk according to McAfee classification. Medium and high risk are quite small and account for less than 1%.

With respect to content *type*, no (broad) category dominates the traffic. A big difference among DVPN arises when considering pornography, which accounts for 27% and 12% of the Mysterium and Sentinel traffic respectively, while it only accounts for 2% of the Tachyon traffic. Only a minority of traffic falls into the *malicious* category (less than 2%). In this category, "illegal software" is the most popular sub category, followed by "phishing" and "malicious websites".

2.4 On the Value of Spare Bandwidth

In this section, I first leverage the previous analysis to derive several assumptions which allow us to solve the optimization problem described in § 2.2.3. Then, I derive optimal buyer's cost and seller's income, and conclude by commenting on the value of spare Internet bandwidth.

2.4.1 Buyer Cost Analysis and Optimization

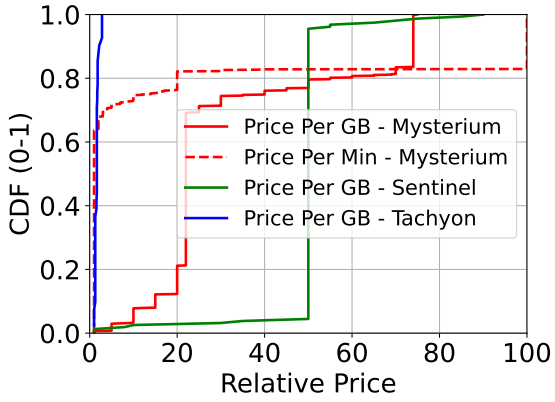
Methodology – I investigate a buyer's minimal cost following Equation 2.2 from § 2.2.3. I first need to filter the sellers based on a buyer's *constraints*, and then find the seller asking the lowest price for her bandwidth. I proceed as follows. For a given marketplace, I obtain the list of sellers as reported by my active crawler. Next, I remove the sellers which provide less bandwidth than b,

the bandwidth requested by a user, and are not in the desired locations L. As per Equation 2.2, I should also filter sellers that block access to the set of domains contacted by a buyer. I skipped this step because blocklist usage is not publicly available for any of the DVPNs.[3] Finally, I choose the seller with the lowest cost among the remaining sellers. If the payment scheme of the marketplace is solely based on the amount of data transferred, *e.g.,* Sentinel, then I choose the seller asking the lowest price. If the payment scheme depends on both the amount of data transferred and the connection duration, *e.g.,* Mysterium, I need to consider the bandwidth needs b of the buyer as well. That is, minimizing the objective function $(x \cdot b \cdot u + y \cdot u)$ (Equation 2.2), where x, y are the prices (of data transferred and connection duration) and u is the connection duration.

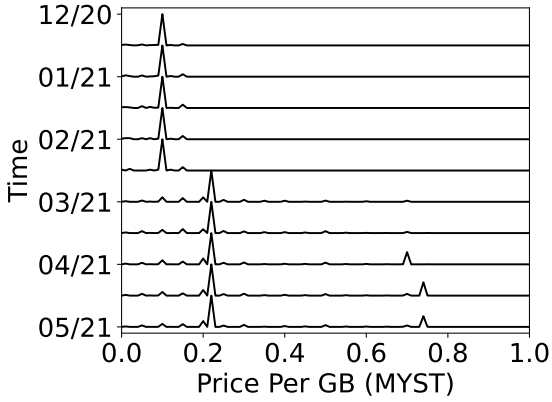
With the existing DVPN apps, the buyers first retrieve the list of all nodes and then manually select a node. By default nodes are ordered by price, *i.e.,* the top of the list shows the cheapest nodes. Hence the above strategy can partially be realized with the current DVPN apps, *i.e.,* the buyers can filter nodes by a given country and manually implement such node selection. However, only "signals" (bad, medium, good) of bandwidth but no exact bandwidth numbers of the DVPN nodes are available in the existing DVPN apps.

Results – Figure 2.6(a) shows the CDF of the seller's prices available to buyers across marketplaces. Prices refer to a sample collected by the active crawler (05/01/2021) and are normalized relative to the lowest price allowed by a DVPN, *e.g.,* given the lowest price for Sentinel is 1 SENT per Gigabyte, then a relative price of 50 indicates 50 SENT per Gigabyte. The figure shows that 40% of Tachyon sellers ask the minimum price (0.22 IPX/GB), and only few (10%) dare to increase the price to 3x the minimum (up to 0.62 IPX/GB). Conversely, most Mysterium and Sentinel sellers (50-90%, respectively) request the default price (50 SENT and 22 MYST), while the remaining

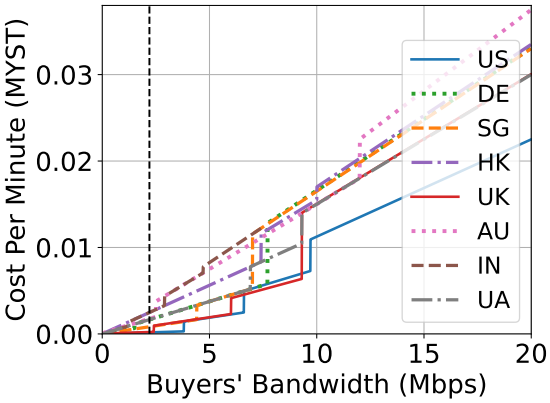
[3]While I could actively test whether a DVPN node block some high-risk traffic, I opted to avoid this experiment for two reasons. First, it would provide a very coarse approximation of existing blocklists. Second, it involves injecting high-risk traffic, which is unethical and potentially illegal.



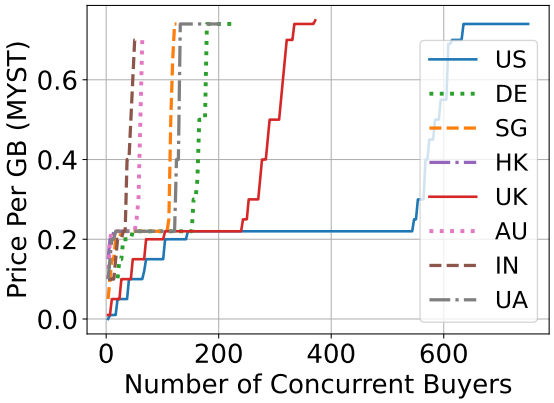
(a) CDF of relative price (to the lowest) per marketplace.



(b) 6-months PDFs of Mysterium prices.



(c) Buyer's minimal cost in Mysterium given a target country and bandwidth.



(d) Mysterium's price per GB as a function of the number of concurrent buyers.

Figure 2.6: Price and buyer's cost analysis.

sellers equally split between providing *much* lower or higher prices (70x for Mysterium and 85x for Sentinel). It is worth noting that Mysterium sellers treat "price per min" differently from "price per GB", mostly requesting the minimum price. This behavior is an indication that complex pricing schemes can be hard to grasp by sellers.

The above result suggests that both Tachyon and Sentinel are not yet realistic bandwidth mar-

ketplaces, despite their large footprint (see Figure 2.2(a)). In fact, their sellers just ask either the minimum (Tachyon) or the default (Sentinel) price unmotivated by the lack of payment as well as cost for the buyers. In the rest of the analysis I only focus on Mysterium since it is currently the most mature bandwidth marketplace.

I start by analyzing the *history* of Mysterium's seller prices. Figure 2.6(b) shows the historical frequencies (PDF) of MYST per GB over 5 months. Mysterium had a major update in late February 2021 when the default price increased from 0.1 to 0.22 MYST per GB, as indicated by the PDF shift in Figure 2.6(b). While many sellers (48%) stick to the default price setting, the number of sellers offering cheaper prices, comprised between 0.01 and 0.22 MYST, grew by 11% and currently account for 21% of the sellers. Further, the figure shows a new peak around 0.7 (04/2021) and 0.74 MYST (05/2021) which account for about 18% of sellers.

I now investigate the question: *what is the minimum buyer cost?* I assume Mysterium buyers with variable bandwidth requirements (between 1 and 20 Mbps, which is mostly an upper-bound as per Figure 2.4) and interest in several countries. I choose countries whose trend in price setting is representative of most other countries in their *regions*, *e.g.,* Germany (DE) for Europe. I assume a buyer can always select the seller which meets his/her constraints, at the minimum price. I will relax this constraint later.

Figure 2.6(c) shows the minimal buyer cost as a function of both bandwidth and location. I express the buyer cost as *cost per minute*, to incorporate in a single metric both pricing schemes adopted by Mysterium. The cost per minute is the sum of the cost of the GB transferred in a minute, given a target bandwidth, and the cost for such duration. The figure shows a significant impact of the selected country on a buyer's cost: up to 10x when comparing the cheapest country (US, UK) with the most expensive one (India). Given that many countries offer high bandwidth, there is potential of savings for buyers who are not interested in a specific location, *i.e.,* they leverage a

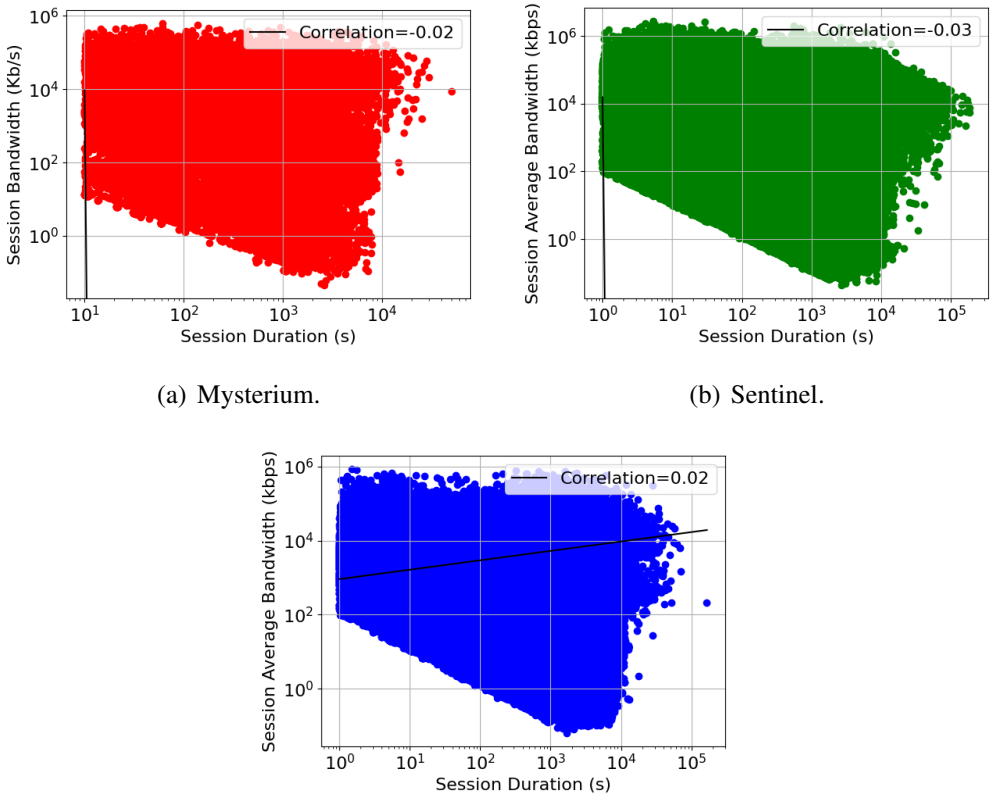
dVPN mostly for privacy.

When focusing on bandwidth requirements, Figure 2.6(c) shows that the cost mostly grows linearly as the bandwidth increases. This is simply because higher bandwidth requires more data per minute. However, I also observe some non-linear "jumps". For instance, the minimum cost for acquiring less than 3.7 Mbps in the US is 0.01 MYST/GB + 0.00001 MYST/min. When higher bandwidth is requested (between 3.7 and 6.6 Mbps) the minimum price available is 5x higher, or 0.05 MYST/GB + 0.00005 MYST/min. Similar patterns apply to other countries, where there are some cheap sellers with relatively small bandwidth capacity, while it costs more to acquire higher bandwidth. Some countries have relatively low bandwidth offerings, *e.g.,* the highest bandwidth provided in India (IN) is only 7.6 Mbps.

Next, I assume N concurrent buyers. Each new buyer consumes a portion of the available bandwidth at a seller (see Figure 2.2(c)), and will thus eventually impact the decision of future buyers. For this analysis, I assume each buyer requires 2.2 Mbps (average peak bandwidth from the distribution described by Figure 2.4(c)). Figure 2.6(d) shows, for several locations, the minimum cost for the N-th user as a function of N, or the number of concurrent buyers. As the load on the marketplace increases, new buyers are left with more expensive sellers, and thus with a bill which grows, overall, by 70x. The cost increases faster in countries with overall less bandwidth for sale. For example, 100 concurrent buyers are enough to force new buyers to pay 0.7 MYST per GB in SG and UA. Conversely, the US can support up to 600 concurrent buyers before reaching such a high price.

2.4.2 Seller Income Analysis and Optimization

Methodology – I investigate a seller's maximum income following Equation 2.4 (§ 2.2.3). For the same reason as in § 2.4.1, I ignore the impact of blocklists. It follows that to solve the objective



(a) Mysterium.

(b) Sentinel.

(c) Tachyon

Figure 2.7: Correlation between session throughput and session duration for (a) Mysterium, (b) Sentinel, and (c) Tachyon.

function I only need to calculate the expectations of the traffic volume $\mathbb{E}[n \cdot B \cdot U]$ and duration $\mathbb{E}[n \cdot U]$. I approximate the distributions of the variables (n, B, U) by their best fitting functions over the data I have collected. For example, the distribution of the DVPN session duration (Figure 2.4(a)) is fitted by function $y = \frac{1}{ax^b+c}$.

I now solve the seller's optimal income as defined by Equation 2.4 (§ 2.2.3). To solve the objective function, I need to: a) calculate the expectation $\mathbb{E}[\mathbf{n} \cdot \mathbf{B} \cdot \mathbf{U}]$ and duration $\mathbb{E}[\mathbf{n} \cdot \mathbf{U}]$, and b) quantify the impact of the blocklist on bandwidth demand. I start off by deriving assumptions based on the measurements.

First, I observe that the session duration is not correlated to the session throughput in any form as illustrated in Figure 2.7, where the throughput spans all values for a given session duration. Next, I notice that the session duration is not affected by implementing a bandwidth limit r as well. The independence is verified using the Kolmogorov–Smirnov (KS) test [240]. The null hypothesis is that there is no difference between the two distributions of the session duration. Given the size of the sampled data (number of sessions), the null hypothesis is accepted when the D-statistics is below 0.021, 0.011, 0.008 given significance level $\alpha = 0.001$ for Mysterium, Sentinel, and Tachyon, respectively. As shown in Figure 2.8, the null hypothesis is accepted in most cases, that is, the session duration **U** is independent from the bandwidth limit r and thus the number of buyers **n**.

Based on these empirical observations, I make the following assumption: the session duration **U** is independent from both the number of buyers **n** and the buyers' throughput **B**, which implies

$$\begin{cases} \mathbb{E}[\mathbf{n} \cdot \mathbf{B} \cdot \mathbf{U}] = \mathbb{E}[\mathbf{n} \cdot \mathbf{B}] \cdot \mathbb{E}[\mathbf{U}] \\ \mathbb{E}[\mathbf{n} \cdot \mathbf{U}] = \mathbb{E}[\mathbf{n}] \cdot \mathbb{E}[\mathbf{U}] \end{cases} \tag{2.5}$$

Let $f(b)$ and $g(u)$ be the PDFs of **B** and **U**, respectively. Following my assumption that the duration does not depend on other variables, the expectation of the duration is a constant:

$$\mathbb{E}[\mathbf{U}] = \int_0^\infty u g(u) du. \tag{2.6}$$

Given the results from Figure 2.4(d), the fraction of number of sessions can be expressed as a function of a node's available bandwidth, which I describe as $H(r)$ and model according to Figure 2.4(d). Then I have:

$$\mathbb{E}[\mathbf{n}] = H(r) \cdot N(x, y, l, A), \tag{2.7}$$

Bdw Limit	1 (Mbps)	5	10	15
5	.012			
10	.011	.009		
15	.010	.014	.012	
25	.020	.008	.013	.020

(a) Mysterium.

Bdw Limit	1 (Mbps)	5	10	15
5	.0059			
10	.0044	.0040		
15	.0088	.0058	.0059	
25	.0081	.0076	.0046	.0077

(b) Sentinel.

Bdw Limit	1 (Mbps)	5	10	15
5	.0025			
10	.0040	.0022		
15	.0057	.0038	.0029	
25	.0069	.0046	.0017	.0014

(c) Tachyon.

Figure 2.8: KS-test statistics of session duration between different bandwidth limits for (a) Mysterium, (b) Sentinel, and (c) Tachyon.

Further, the volume expectation is the product of the number of buyers with lower throughput than the bandwidth limit ($b \leq r$) and the mean throughput of these buyers:

$$\mathbb{E}[\mathbf{n} \cdot \mathbf{B}] = H(r) \cdot N(x, y, l, A) \int_0^r bf(b)db. \tag{2.8}$$

As discussed in § 2.4.1, I ignore the impact of blocklists since they are not publicly available for any DVPN. This leaves us with the following objective function for seller's optimal income:

$$\begin{aligned} &\max_{x,y,r} \mathbb{E}[\mathbf{U}] \cdot H(r) \cdot N'(x, y, l) \cdot (x \int_0^r bf(b)db + y) \\ &s.t.\ \ \mathbb{E}[\mathbf{U}] \cdot H(r) \cdot N'(x, y, l) \int_0^r bf(b)db \leq D/2 \end{aligned} \tag{2.9}$$

Let us fix the location and ignore the blocklist, and denote $\int_0^r bf(b)db$ by $F(r)$. Then the

objective function of the sellers is

$$\begin{aligned} &\max_{x,y,r} \mathbb{E}[\mathbf{U}] \cdot H(r) \cdot N"(x,y) \cdot (x \cdot F(r) + y) \\ &s.t.\ \ \mathbb{E}[\mathbf{U}] \cdot H(r) \cdot N"(x,y) \cdot F(r) \leq D/2 \end{aligned} \tag{2.10}$$

The solution is as follows. Let $C(x, y, r)$ be the constraint function

$$C(x,y,r) = \mathbb{E}[\mathbf{U}] \cdot H(r) \cdot N"(x,y) \cdot F(r) - D/2. \tag{2.11}$$

And let

$$\begin{aligned} \mathbb{L}(x,y,r) =& \frac{1}{\mathbb{E}[\mathbf{U}]} \cdot [I(x,y,r,l,A) - \beta C(x,y,r,l,A)] \\ =& H(r) \cdot N"(x,y) \cdot (x \cdot F(r) + y) \\ & - \beta (H(r) \cdot N"(x,y) \cdot F(r) - \frac{D}{2}) \end{aligned} \tag{2.12}$$

I have

$$\begin{aligned} \nabla_{x,y,r,\beta} \mathbb{L} =& H(r) \cdot \{\frac{\partial N"}{\partial x}[(x-\beta)F(r) + y] + N" \cdot F(r)\}\partial x \\ &+ H(r) \cdot \{\frac{\partial N"}{\partial y}[(x-\beta)F(r) + y] + N"\}\partial y \\ &+ N" \cdot \{\frac{\partial H(r)}{\partial r}[(x-\beta)F(r) + y] \\ &+ r f(r) H(r)(x-\beta)\}\partial r \\ &+ \{H(r) \cdot N" \cdot F(r) - \frac{D}{2}\}\partial \beta \end{aligned} \quad , \tag{2.13}$$

which, in accordance to the Lagrange multiplier theorem, gives the optimal condition for maxi-

mizing the objective as

$$\begin{cases} \dfrac{\partial N"}{\partial x}[(x-\beta)F(r)+y]+N" \cdot F(r)=0 \\ \dfrac{\partial N"}{\partial y}[(x-\beta)F(r)+y]+N"=0 \\ \dfrac{\partial H(r)}{\partial r}[(x-\beta)F(r)+y]+rf(r)H(r)(x-\beta)=0 \\ H(r)\cdot N" \cdot F(r)-\dfrac{D}{2}=0 \end{cases} \tag{2.14}$$

Results – Figure 2.9 shows the average number of sessions (black markers) and total income (orange markers) they produce for Mysterium sellers under variable pricing and bandwidth limits. Each point further shows minimum and maximum value of each metric as errorbars. The dashed lines represent a fitting function of daily sessions ($N(x) = ae^{x-b} + c$, black) and the theoretical income expectation (Equation 2.9, orange). These results are derived from a one-month experiment where I fix the location l to be the US, set no data cap $D = \infty$, and allow all traffic types. To bound the number of variables, I investigate 6 different prices per GB (0.01, 0.02, 0.05, 0.1, 0.22, and 0.5) MYST but fix the price per minute to the minimum, *i.e.,* the most popular option as suggested by Figure 2.6(a). For the nodes bandwidth I select 5 and 25Mbps, which are *low* and *average* bandwidth currently offered by Mysterium nodes in the US.

In presence of high bandwidth (Figure 2.9(a)), the figure shows that the number of daily sessions quickly decreases as the price increases. For example, a 10x price increase (from 0.01 to 0.1 MYST per GB) causes the average number of daily sessions to drop from 44 down to about 4 sessions, and then eventually near zero as the price keeps increasing. This behavior causes the income to be fitted by a non-linear function: the income grows when the price doubles from the minimal (from 0.01 to 0.02 MYST per GB), then decreases (between 0.02 and 0.22 MYST per GB), and finally flattens out (between 0.22 and 0.5 MYST per GB). A similar trend is observable

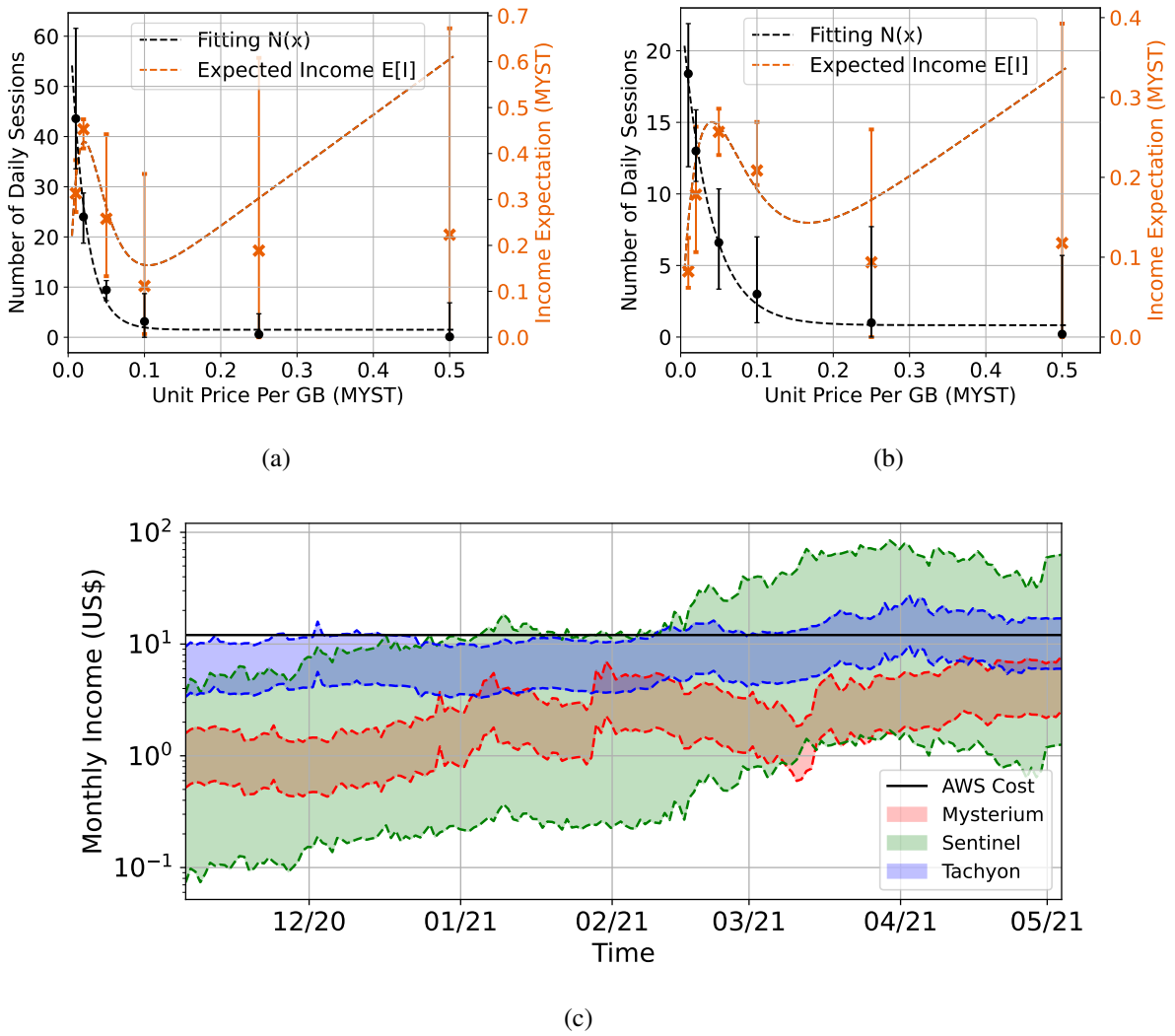


Figure 2.9: Mysterium's seller income analysis as a function of price settings assuming a node bandwidth of: (a) 25Mbps and (b) 5 Mbps. (c) Evolution over six months of default and optimized monthly income per marketplace.

also when I limit the node bandwidth to just 5Mbps. In this case, the node attracts less than half of the sessions but it is also less penalized by a price increase, *i.e.,* peaking at 0.05, or twice as much as before. It follows that, at its peak value, the daily income amounts to 0.28MYST versus 0.45MYST with 25Mbps, or just a 40% reduction in presence of an 80% bandwidth reduction. This

result can be due to many reasons, but mostly indicates that a small portion of Mysterium buyers tend not to optimize their costs, either because they purposely pay more for better experience or they do not optimize their node selections, *e.g.,* filter their nodes by pricing and bandwidth.

I now focus on the accuracy of my theoretical income expectation. The figure shows that when the price per GB is equal or lower than 0.1 MYST, the theoretical results fit the measurement results quite well. However, theoretical and actual incomes diverge when $x > 0.1$, *e.g.,* the theoretical income overestimates the actual income by 3x, compared to the average, with a price of 0.5 GB per MYST. This is because at this price point there are mostly no sessions per day, with the exception of few days which cause high variance, *e.g.,* between 0 and 0.7 MYST with a price of 0.5 GB per MYST and 25 Mbps (Figure 2.9(a)). Due to such high variance, the fitting function for N(x) is quite coarse in this range.

When comparing the latter result with the current pricing adopted by Mysterium sellers (Figure 2.6), I find that most sellers are making suboptimal decisions with respect to their income optimization. Figure 2.9(c) shows, for each marketplace, the lowest (lower bound) and optimized (high bound) incomes over 6 months. Each monthly income is derived using the same strategy as in Figure 2.9(a); I then derive income value in dollars based on the cryptocurrency value during each day of the month, with the goal to visualize its high volatility. The figure shows that a careful mechanism to optimize seller's price offers significant income increases, comprised between 3x (Mysterium) and 50x (Sentinel). However, even assuming optimal income, the monthly income is mostly below $10 (both over time and across DVPNs) or the cost of renting an AWS instance to act as a DVPN node. However, if the increasing trend of cryptocurrency values continue, hosting a DVPN may become more and more profitable. Similarly, it seems that Mysterium has realized that the default pricing scheme is currently too low, as suggested by the change in the default price as observed in Figure 2.6(b). The figure also shows that the volatility of cryptocurrency can make

Table 2.2: The value of spare Internet bandwidth in the US.

	Time (Hr)	Current MYST ($)	Optimized MYST ($)
Buyer's Cost	Average	0.18 (0.11)	0.02 (0.012)
	Most Popular	-	0.026 (0.016)
	Least Popular	-	0.015 (0.009)
Seller's Price	Average	0.23 (0.14)	0.028 (0.017)

one marketplace more profitable than another, over time. For example, Mysterium has been filling the (income) gap with respect to Tachyon over the period of my measurement.

It has to be noted that the above price optimization only applies to a precise period of time. In reality, the optimal price is not fixed but should be evolving over time, influenced by the instantaneous choices of both the buyers and the sellers, as well as the presence of alternative marketplaces. This motivates me to build a system, in the upcoming section, which is capable of adjusting seller's "settings", *e.g.,* pricing and traffic per marketplace, to optimize their income over time.

2.4.3 Discussion

I finally comment on the value of spare Internet bandwidth leveraging the collected data and proposed modeling. I focus on Mysterium – since it has shown to be a mature bandwidth marketplace – and the US, which is currently the largest market (see Figure 2.2(b)) and the location where I conducted my pricing and bandwidth experiments (see § 2.2).

By the end of my measurements, Mysterium counts 302 sellers (Figure 2.2(b)) in the US asking an average price of 0.23 MYST/GB. Using function N(x) from Figure 2.9(a), *i.e.,* assuming 25 Mbps or the average bandwidth offered by nodes in the US, I estimate that these nodes currently attract, in total, 629 daily buyers, that spend between 0.01 and 0.74 MYST/GB (0.18 MYST/GB

on average).[4] It follows that the value of spare Internet bandwidth (in the US) lies between 0.18 MYST/GB (average price paid by the buyers) and 0.23 MYST/GB (average price requested by the sellers), which corresponds to $0.11-0.14 per GB, given the cryptocurrency value then.

Next, I explore the effect of the optimization of buyer's cost and seller's income on the value of spare bandwidth, independently. Figure 2.4(a) shows that the average buyer session lasts about one hour. Assuming 629 daily buyers, equally distributed throughout the day, then there are, on average, 25 concurrent buyers interested in a US node at any point in time ($629 \times 3,499s/24h \approx 25$). I then apply the methodology described in § 2.4.1 (Figure 2.6(d)), where each buyer selects the cheapest seller who has enough spare bandwidth to satisfy her demand. After this optimization, the buyers will pay between 0.01 and 0.05 MYST/GB (0.02 MYST/GB, on average), or a 10x reduction compared to today (0.18 MYST/GB). I then take a closer look at the time of the day. My measurements indicate that there are 33 and 18 concurrent buyers during the most and least popular hour, respectively. Table 2.2 shows the optimized buyer's costs in these scenarios. From the seller's perspective, I have previously derived optimal price setting of 0.028 MYST/GB (Figure 2.9(a)), which is 8x smaller than the average price (0.23 MYST/GB) that the sellers are requesting today. Note that the table does not show optimal seller's price at different times of the day since I did not observe significant shift. These discrepancies suggest that neither the buyers nor the sellers are optimizing their costs/incomes.

From the above analysis, I conclude that the value of spare US Internet bandwidth lies between $0.11 and $0.14 per GB. However, both buyers and sellers have room to move this price and optimize either their cost or income. This result was obtained considering the optimization of a

[4]More specifically, from the active measurements, I know the number of sellers M given any price x ranging from 0.01 to 0.74 MYST/GB. For instance, $M(0.01) = 4$ means that there are 4 sellers offering at 0.01 MYST/GB. I also know the number of buyers N given a price x per day. For instance, $N(0.01) = 43$ means that a seller with 0.01 MYST/GB attracts 43 buyers per day. Let T_s be the average buyers' session time. I then can estimate the number of concurrent buyers in the US by $(M(0.01) * N(0.01) + M(0.02) * N(0.02) + \ldots + M(0.74) * N(0.74)) * T_s/24h$.

buyer's cost or seller's income, in isolation. In reality, one would affect another. For example, the value of bandwidth would decrease if more sellers optimize their incomes because the bandwidth demand is currently less than the supply, *i.e.,* I estimate 25 concurrent buyers when the sellers can support over 600 (Figure 2.6(d)). Further analysis would require many other assumptions, *e.g.,* rationality of the buyers and sellers, and is out of the scope of this project.

2.5 RING: One DVPN To Rule Them All

This section translates results from the previous section in a concrete system, *RING*, which helps sellers to enhance security protection as well as *maximize* their income while participating to multiple bandwidth marketplaces. I start with a quick extension of the bandwidth monetization problem in the context of a multi-vendor marketplace. Next, I detail design and implementation of RING. I then conclude the section by showing how RING operates.

2.5.1 Multi-Vendor Bandwidth Market Optimization

Consider a seller who joins M marketplaces concurrently. For a marketplace $i \in [1, M]$, I denote by B_i and U_i the probability density functions of bandwidth and duration characterizing its buyer sessions. Next, I denote by x_i and y_i the prices for traffic volume and session duration, per marketplace i. Finally, I call r_i the maximum bandwidth allowed per marketplace and D the *total* data cap, *i.e.,* the sum of all data caps D_i per marketplace. The objective function from Equation 2.4 becomes a cross optimization of the total income from multiple marketplaces (see Equation 2.15):

$$\begin{aligned} &\max_{x_i, y_i, r_i} \sum_{i}^{M} (x_i \cdot \mathbb{E}[\mathbf{n}_i \cdot \mathbf{B}_i \cdot \mathbf{U}_i] + y_i \cdot \mathbb{E}[\mathbf{n}_i \cdot \mathbf{U}_i]) \\ &s.t. \ \sum_{i}^{M} \mathbb{E}[\mathbf{n}_i \cdot \mathbf{B}_i \cdot \mathbf{U}_i] \leq D/2 \end{aligned} \tag{2.15}$$

Algorithm 1: Sellers' Income Optimization Heuristic

Input : $DVPNs[1, M]$. For $i \in [1, M]$, Rate Limit r_i, Price x_i, Data Caps D, D_i, Income I_i, Cumulative Consumed Data CC_i, Last Consumed Data LC_i, Left Time T_{left}.

1 $I \leftarrow \sum_i^M I_i$
2 **for** $i \in [1, M]$ **do**
3 $\quad D_i \leftarrow D_i + \alpha(D\frac{I_i}{I} - D_i)$ // Adjust the data cap
4 $\quad$ Data demands $d_i \leftarrow LC_i \cdot T_{left}$
5 $\quad$ **if** $d_i > D_i - CC_i$ **then**
6 $\quad\quad$ Decrease r_i and/or increase x_i // decrease demands
7 $\quad$ **else**
8 $\quad\quad$ Increase r_i and/or decrease x_i // increase demands
9 $\quad$ **end**
10 **end**

Output : Rate limits r_i and price settings x_i.

In reality, it is challenging to collect the information needed to solve Equation 2.15. Based on the insights I have gained from the optimization of the seller's income in a single marketplace (see § 2.4), Algorithm 1 proposes a local heuristic to approximate the solution of the above optimization.

First, I calculate total (across marketplaces) income for all DVPNs (L1 of the algorithm) in a time interval T, *e.g.,* one hour. The interval should be neither too short, since changing the settings requires rebooting the DVPN (thus interrupting all ongoing sessions), nor too long which slows down algorithm's convergence to the optimal settings. I have tested several time intervals and found that one hour is appropriate. The total data cap D is a user provided constant, which is initially equally distributed among marketplaces (D_i). Each marketplace may generate different income due to, for instance, the current value of its cryptocurrency. I thus adjust the data cap for each marketplace to maximize the income. When a marketplace has generated more income per GB than the average income per GB for all marketplaces ($\frac{I_i}{D_i} > \frac{I}{D}$), this is an indication that its data cap D_i should be increased. Otherwise its data cap should be decreased. I iterate across

marketplaces and adjust their data cap (L4), where the *aggressiveness* of data cap reallocation is determined by coefficient α, *i.e.,* when $\alpha = 1$, the data cap is adjusted purely based on what happened in the last hour.

Next, I adjust the bandwidth limits and/or prices to maximize the profit for each marketplace. § 2.4 suggests that the key to maximize a seller's income is to adjust bandwidth limit and price such that the buyers' bandwidth demand fulfills the seller's data cap. I calculate the buyers' bandwidth demand based on consumed data in the last hour (L5). Then, I compare the derived bandwidth demand with the leftover data cap (L6-L10). If the bandwidth demand exceeds the data cap, then I either increase the price or decrease the rate limit if the marketplace allows charging for session duration (*i.e.,* $y_i > 0$). In fact, according to § 2.4 this would allow to reduce the bandwidth demand and increase income. Otherwise, I either decrease the price or increase the rate limit to attract more buyers, and thus increase income.

2.5.2 Design and Implementation

RING's design is motivated by four goals. First, allow a seller to *concurrently* join multiple marketplaces (DVPNs). Second, provide intelligence to *maximize* a seller's income. Third, provide *fine-grained* control on permitted traffic to limit the danger of running a DVPN node. Fourth, *ease of use*: currently, mostly expert users can deploy DVPN nodes due to lack of executables across OSes, complex setup, etc.

Figure 2.10 shows RING's architecture with its three main components: *client*, *manager server*, and *crawler*. The crawler is the same one described in § 2.2: it periodically crawls the set of supported DVPNs to fetch information like available nodes, and current pricing. The client controls and monitors multiple DVPN nodes running at the user machine. It fetches up-to-date information about the DVPNs and makes decisions for local bandwidth allocation and price settings. Below, I

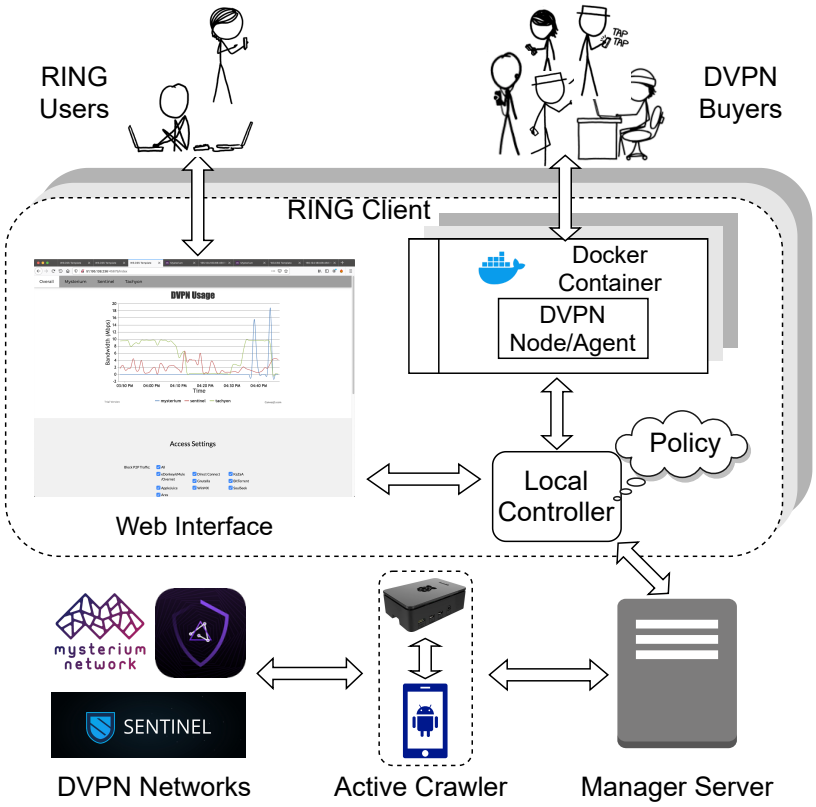


Figure 2.10: Visual representation of RING.

provide details on RING's client and its traffic control.

RING Client – I adopt Docker containers [54] which allow to run DVPN nodes concurrently and in isolation. I create Docker virtual network interfaces which eases traffic monitoring and rate limiting per DVPN. I build Docker images from each DVPN up-to-date source code to support Raspberry Pi (ARM), which I envision as the perfect plaform for RING's clients – a small and cheap box to attach to the home router.

RING's client can be managed by a Web interface. This interface makes it possible to customize each DVPN, *e.g.,* by providing crypto-wallet addresses, speed limits, data cap, and allowlists. Further, the interface shows several useful statistics, *e.g.,* bandwidth consumed by each

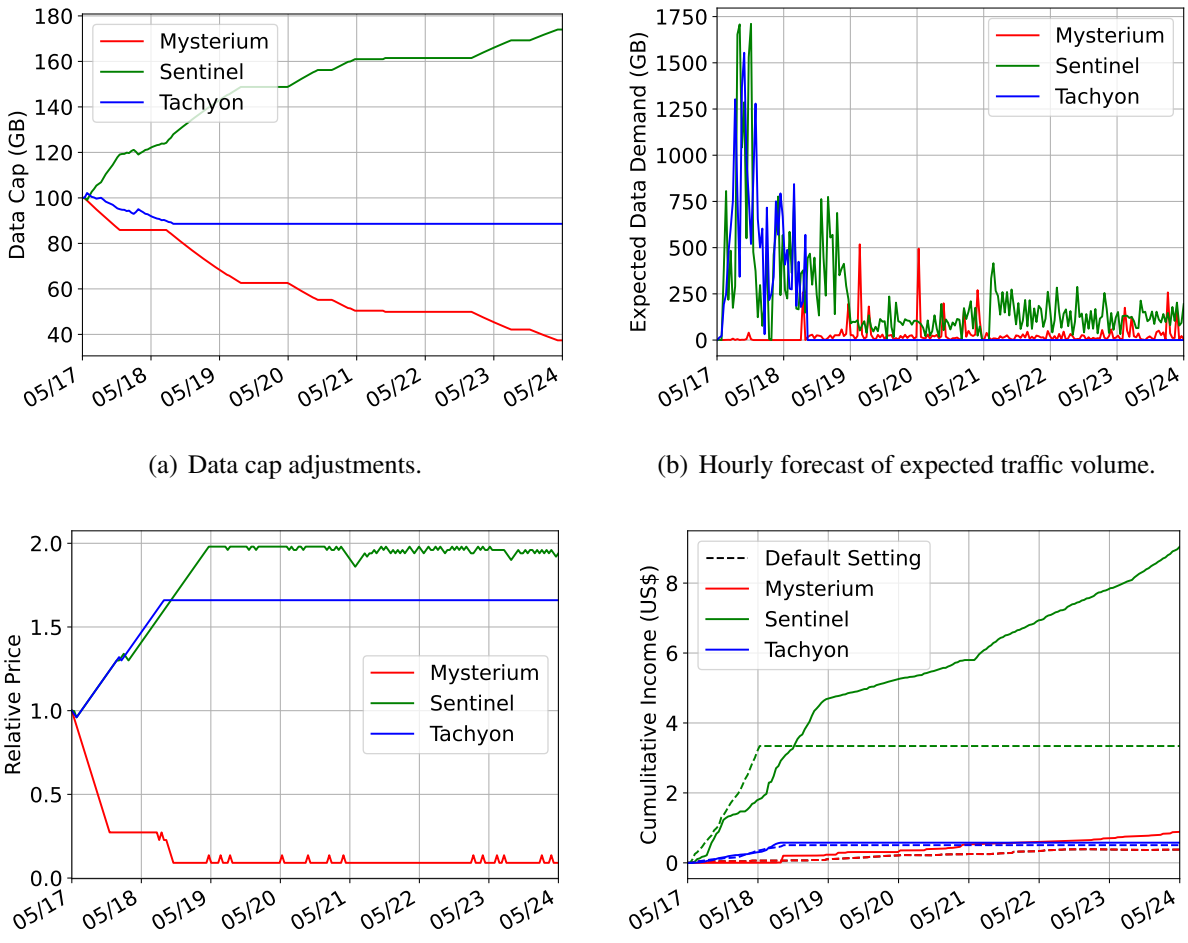
DVPN as shown in Figure 2.10. The Web interface (locally) communicates with a controller which executes user input, *e.g.,* starts/pauses/stops a specific DVPN or updates the price settings. RING's client can also set timer and choose specific times of the day to be a part of the marketplace. Other important user inputs are *rate limits*, which translate into Linux TC [36] rules, and *accesslists*, which translate into iptables. Next, I offer more details on how accesslists are implemented.

Traffic Control – RING's Web interface allows users to only allow low, medium, high risk traffic [6], or select a set of content categories allowed, *e.g.,* avoid gambling and pornography. An option to block P2P traffic – identified using IPP2P – is also provided. To generate such iptables rules, I rely on the domain classification described in § 2.2: SNI + DNS data matched using McAFee database [6] which achieves, in my data-set, >95% domain coverage. Such rules are maintained at the manager and updated regularly.

2.5.3 Evaluation

I demonstrate the functioning of RING via controlled experiments (over one week) using two AWS EC2 machines in the US: one machine running RING's heuristic and one using default prices and no control on data cap and bandwidth limit. I assume a 300 GB weekly data-cap, initially equally divided between the three DVPNs. I derive income for each controlled user based on the amount of traffic they carried, their price settings (either default or using RING price adjustments), and the cryptocurrency value. In the case of Mysterium, I also validate such computed income against the official figures reported by Mysterium. This cannot be done for Sentinel and Tachyon which are still in development and do not yet release payments to their users. In the following, I first discuss the decisions made by RING's heuristic and then compare the income generated by the two machines.

Figure 2.11(a) shows the evolution over one week of the data cap allocation per DVPN realized



(a) Data cap adjustments.

(b) Hourly forecast of expected traffic volume.

(c) Price decisions.

(d) Cumulative income.

Figure 2.11: RING's preliminary evaluation over one week: (a) data cap adjustments, (b) hourly forecast of expected traffic volume, (c) price decisions, and (d) cumulative income.

by RING, which is automatically adjusted based on the computed income per GB. The figure shows that Sentinel's data cap almost doubles over one week, from the starting 100 GB up to 170 GB, at the expense of Tachyon (down to 90 GB) but mostly Mysterium (down to 40 GB). This implies that Sentinel is bringing the most income per GB, as previously shown in Figure 2.9(c).

Figure 2.11(b) shows a hourly forecast of the expected traffic volume for the week given the

rate measured in the last hour. The figure shows that, within two days, both Sentinel and Tachyon attract significant amounts of traffic which would exceed their weekly data caps. Their prices are then increased (Figure 2.11(c)) which effectively reduces the demand (Figure 2.11(b)) while increasing income (Figure 2.11(d)). The opposite behavior is observed for Mysterium; as previously discussed, Mysterium's default price is too high and a price reduction can attract more traffic (see Figure 2.9). Because I adjust price and bandwidth limit conservatively, Tachyon runs out of data cap on the second day. With appropriate price setting, Mysterium and Sentinel show data demands (50 and 150 GB) that closely approximate their data caps.

To conclude, Figure 2.11(d) shows the cumulative income generated using both default settings for each DVPN, and RING's heuristic. Because of the high demand shown in Figure 2.11(b), Sentinel and Tachyon with default settings run out of data cap on the second and third day respectively, generating a total income of $3.7. In contrast, RING allocates a larger data cap to Sentinel and also increases its price, generating a combined (Sentinel plus Tachyon) income of $10.5. For Mysterium, RING's price adjustment allows to double its income compared to the default setting ($0.9 versus $0.4).

This preliminary evaluation shows that RING achieves higher income than default DVPNs settings. It is noteworthy that the evaluation only reveals partial capability of RING. With the flexibility to deploy more advanced optimization algorithms, *e.g.,* reinforcement learning schemes [129, 130, 131, 203, 305], and the ability to include more DVPNs, RING has a high potential in benefiting the bandwidth sellers. I have released RING [30, 31] to the public.

2.6 Related Work

Several research studies quantify the presence of spare bandwidth. In particular, a measurement study showed that a typical U.S. household does not use most of its bandwidth while streaming

and gets marginal gains from upgrading speeds [39, 116]. Despite such findings, user studies have shown that reliability and speed are most important for consumers [268, 275]. This inclination towards high speeds, enhanced by the proliferation of bandwidth-hungry applications, further impacted by the necessary variability in traffic demand, leaves often significant portions of bandwidth unused. Hence, monetization opportunities, which I analyzed in this project, arise.

Data caps are a method ISPs use to protect against heavy-hitters [48] or to limit user activity in resource-constrained networks such as cellular networks [124]. There has been research on the implications of data caps on the user experience. For example, in [138] the authors explored the effects of data caps on home Internet usage in urban South Africa to show that users have three uncertainties with regards to their bandwidth usage: invisible balances, mysterious processes, and multiple users. My project considers a different scenario, the one where dVPN nodes operate under data caps, yet because these caps are unlikely to be reached, they monetize spare bandwidth. ISPs are faced with a tradeoff – make their plans less attractive by reducing data caps and negatively affect users and their own revenues, or enable larger caps to attract users and consequently enable bandwidth monetization.

There has been research on understanding the users that are willing to pay more for bandwidth. Necessarily, such a willingness is positively related to income and other technological attributes and negatively related to socio-demographic attributes such as habitat and age [126]. Another study found that there exists a significant variability in the sense that certain kinds of users are willing to pay substantially more than others [297]. Such a variability exists with dVPNs, which are currently in their "infancy." Hence, there is a narrow group of consumers involved in the spare bandwidth market. Nonetheless, this market is likely to grow in the coming years and become mainstream. In this project, I analyzed the key parameters that affect this emerging market.

There has been work on addressing bandwidth pricing among users and ISPs considering

single- and multi-class scenarios [283]. Others have analyzed incentives for creating efficient *inter-ISP* bandwidth marketplaces [125] or pricing schemes among different to accomplish net neutrality [180]. My work is distinctive because I effectively have Internet consumers both as buyers and sellers of the spare bandwidth market. A more similar scenario is the P2P system BitTorrent, where Internet users trade the download resources with upload capacity, *i.e.,* bandwidth for bandwidth. There has been work on studying the incentives of users in such a P2P system [221, 256]. Yet, my investigated market is different because the trading resources are not the same, *i.e.,* bandwidth trading with money, and the latter resource depends on geolocation, *i.e.,* purchasing power is different in different countires, and is potentially unlimited with respect to my investigated market. Also, unique issues affect such a market, *e.g.,* seller's location and willingness to serve a particular type of traffic.

Network censorship, *i.e.,* blocking traffic originated to and from particular applications or regions [108], is one of the main drivers behind the consumer-consumer bandwidth marketplace I explored in this project. Typically, buyers come from censored regions and sellers reside in the remainder of the Internet. My data (details omitted) confirm that this is indeed the case.

2.7 Discussion

Ethical Consideration – My work involves human subjects, *i.e.,* users who connect to dVPN nodes hosted by us. I followed the best community practices when conducting my work to make my data collection anonymous. Two identifiers are available for DVPN users: IP addresses and dVPN-specific identifiers. I perform coarse-grain geo-location analysis on the IP addresses which contact my nodes and then discard them. Further, dVPN-specific identifiers are not exposed to my nodes for both Sentinel and Tachyon, and in case of Mysterium I do not record them. IRB at Northwestern University has determined that my work is not considered human research because

I used non-identifiable private information about living individuals and data collected does not contain any accompanying information by which I could identify such individuals.

Measuring Bandwidth Marketplace With Cryptocurrency Transactions – This project focuses on measuring the DVPNs based on network activities, such as the initiation of VPN sessions and the amount of data transferred. Given that the DVPNs are based on blockchain, this analysis can be further enriched by examining cryptocurrency transactions recorded on the blockchain. This would allow us to leverage the unique transparency of blockchain technology to understand the economic transactions underpinning the service, shedding light on their operational effectiveness and the economic dynamics driving their usage. I leave such explorations as future work.

Bandwidth Marketplace Dynamics and Cryptocurrency Valuations and Mining – This project explores the bandwidth marketplace, focusing on modeling aspects under the initial assumption that marketplace dynamics remain unaffected by fluctuations in cryptocurrency values or mining activities. However, this assumption oversimplifies the complex interplay between the marketplace and cryptocurrency economics. The reality is that both buyers' pricing strategies and sellers' revenues are intricately linked to cryptocurrency valuations. Moreover, there exists a reciprocal relationship between bandwidth marketplace operations, such as VPN tunnel transactions, and cryptocurrency mining efforts. Mining processes not only facilitate marketplace activities by grouping, verifying, and recording transactions on the blockchain but also influence the economic incentives for miners based on the transaction volumes within the marketplace. Consequently, the decisions of buyers and sellers regarding which DVPNs to engage with are significantly influenced by the state of the cryptocurrency market. To achieve a more detailed understanding, it would be interesting to incorporate the game theoretical dynamics prevalent in cryptocurrency markets [89, 106, 185, 238, 239, 290, 309, 317] into the analysis, which would shed light on the nuanced economic interactions at play.

Security and Privacy Issues – My passive measurement effectively reveals numerous traffic characteristics, including origins, destinations, duration, throughput, and categorization, among others. It has also helped to model the bandwidth marketplace behind DVPNs. Nevertheless, it has also brought potential security and privacy issues of DVPN to light.

First, there is no protection for clients' traffic, leaving it vulnerable to monitoring and interception by DVPN nodes, particularly for plaintext packets like DNS queries and responses. While the adoption of HTTPS traffic can reduce this risk, my study (§ 2.3) still demonstrates what information a DVPN node provider can obtain by analyzing ongoing traffic. Such analysis could lead to privacy breaches for DVPN clients. It is noteworthy that while I refrained from conducting individual-level analysis due to ethical concerns, a malicious DVPN node provider could technically perform such analysis, which could result in serious privacy breaches for DVPN clients. I have found that there are no feasible solutions to this problem without complete control of the entire wild-area network. Indeed, even if one can ensure that the DVPN node does not passively monitor the traffic, it is impossible to guarantee that home routers or other middleboxes along the route do not.

Second, there are also great risks of joining the DVPN as a node provider. Popular DVPN networks like Sentinel and Tachyon provide no option for node providers to protect themselves from dangerous traffic. While Mysterium offers such an option, it does not provide further details on what traffic will be filtered. RING, which I built, enables this protection for node providers. Still, I acknowledge that RING relies on third-party databases that may be subject to attacks due to late updates or false positives. Additionally, a malicious client can locate all IP addresses of DVPN node providers by accessing APIs provided by DVPN networks, *e.g.,* Mysterium and Sentinel, or by continuously restarting new connections if such an API is not available, as with Tachyon. This creates potential privacy breaches and security issues for DVPN node providers.

Lastly, the existing DVPNs are not truly "decentralized", as they often rely on a centralized endpoint at the boot of the service. For instance, Mysterium clients must first access the URL *[testnet-]broker.mysterium.network* before connecting to a DVPN node. This makes the DVPN service vulnerable to attacks at the centralized endpoint and reduces censorship resistance, which is a primary goal for many DVPN clients (see § 2.3).

2.8 Summary

The decentralized VPN is an enhancement to the centralized VPN with respect to the security and privacy of Internet users. Given the increasing residential Internet speeds and the rise of cryptocurrencies which allow easy transactions to be made between any individuals on earth, multiple DVPNs have been proposed and attracted many users. In this chapter, I have presented the first comprehensive study of the ecosystem of DVPNs. I *actively* and *passively* monitored three major DVPNs (Mysterium, Sentinel, and Tachyon) for 6 months, reporting on their footprint, performance, income opportunities, and traffic characteristics. Using this data, I estimated that the value of spare Internet bandwidth in the US ranges between 11 and 14 cents per GB. Still, I found that both buyers and sellers utilize ad-hoc "rules-of-thumb" when choosing their prices, resulting in a sub-optimal marketplace. Indeed, I showed that a seller's income could be increased by setting a lower but optimal price which is likely to attract more buyers. I also predict that the value of spare bandwidth would be reduced when more sellers begin to optimize their income as the current bandwidth supply exceeds the demand. Finally, I formalized how a seller's income could be optimized in a *multi-vendor* marketplace. I also realized this abstraction in RING, the first such marketplace built on top of Mysterium, Sentinel, and Tachyon, which helped increase a node revenue by 63%. RING also enhances the existing DVPNs on the security guarantees.

However, even with the adoption of multiple hops, as suggested by Mysterium and Sen-

tinel [22] or Tor [40, 160], I find that DVPNs are still unable to provide adequate security and privacy guarantees without control of the entire wild-area network and user machines. Ultimately, any form of VPN is bound to some trust assumptions, making it difficult to obtain complete security and privacy guarantees.

CHAPTER 3

PDNS: ENHANCING DNS PRIVACY WITH PRIVATE INFORMATION RETRIEVAL

The Domain Name Service (DNS) is the phonebook of the Internet [45] which maps IP addresses like "151.101.195.5" to human-friendly names like "cnn.com". At the birth of the Web, security and privacy were not contemplated, leaving DNS traffic as plaintext. This means that *any* (middle)box placed between a DNS client and *recursive resolver* (ReR) could monitor user activity, potentially building accurate user profiles [178]. Twenty-eight years later, DNS-over-TLS (DoT) [157] and DNS-over-HTTPS (DoH) [187] solve this limitation by mean of end-to-end encryption. DoT and DoH have been gradually supported both by clients (*e.g.,* browsers like Chromium [84] and Firefox [80]) and ReRs [79, 81].

End-to-end encryption protects a user's privacy from eavesdroppers but not from a ReR. ODNS [277] is a recent solution — already deployed by Cloudflare [285] — to address such problem by detaching a user identify from a DNS request. This is achieved by adding a proxy between DNS client and ReR such that: 1) the proxy is blind with respect to an encrypted DNS query, 2) the ReR is blind with respect to the client's identity (IP address). Assuming a non-colluding proxy and ReR, user privacy is enforced. However, non-collusion is hard to enforce and verify in reality. For example, both proxy and ReR can be subjects of a subpoena, at which point privacy is again sacrificed. Finally, ODNS still allows the ReR to gather knowledge about the users as a whole, *e.g.,* answer questions like *"what is the most popular online newspaper, and its potential political affiliation, in a given region?"*

The only way to protect users from the above privacy infringement would be either removing the ReRs from DNS [278], or having ReRs operating *in the blind*, *i.e.,* by resolving domains

without knowing what they are. The former option exhibits high performance penalties to users, amplifies workloads on the ANSes, and raises additional security concerns. The latter option seems counter-intuitive, but in reality several techniques exist which allow similar operations. These techniques fall in the branch of Private Information Retrieval (PIR), which is achieved by various cryptographic tools such as homomorphic encryption [113, 172, 173, 264, 264]. Indeed, private DNS is often cited as a motivating example in PIR research, but no practical implementation currently exists.

The goal of this work is to fill the gap between PIR and DNS research. I do so by introducing PDNS, a Privacy-Preserving DNS designed to *augment* rather than replace DNS, in a spirit similar to DoH and ODoH. To achieve my vision, I had to solve the following challenges.

PIR Selection and Optimization: Out of all the available PIR categories, I suggest utilizing the single-server stateless PIR schemes for DNS, as they don't require a non-collusion agreement, bear low costs for cache updates, and offer satisfactory running times for query processing. I benchmarked multiple schemes and find that Spiral [241] offers the highest performance. To integrate Spiral into DNS, I researched the optimal DNS cache configuration for PIR, and implemented performance enhancements leveraging multi-threading and low-level instruction support.

Cache Population: PIR protocols assume that a database (or cache in DNS context) is either given or can be privately populated. This is not the case for DNS where the ReR is responsible for populating its cache based on the user request. Clearly, a *blind* ReR cannot perform such operation which should be tackled by the client instead. Still, the client cannot update the ReR cache or it would invalidate the system privacy. I propose EDNS-PR, our own EDNS(0) [150] extension which allows a client to communicate the IP address of its ReR in presence of cache misses, so that an authoritative name server (ANS) can privately populate the ReR's cache.

Security Challenges: The previous construction imposes new security challenges for DNS. At-

tackers can either congest ANSes, or launch *reflection* attacks to congest or poison the cache of ReRs. I leverage the security properties of Spiral with digital signatures to allow ANSes to *validate* cache misses when needed, *i.e.,* when suspecting a potential attack.

I implement a PDNS client and ReR, and extend the popular BIND9 [46] to support EDNS-PR as my own extension of EDNS(0) at the ANS. In my experiments, PDNS answer queries 2x faster than DoH over Tor – a privacy-preserving anonymous network – even on a *large* cache (512MB, up to 13M DNS records). PDNS is also faster than ODoH (208ms versus 272ms) with a *small* cache (64MB, up to 1.6M DNS records), and adds 180ms with a large cache. I envision that the advent of specialized hardware for PIR would reduce PDNS' query duration to 70ms (even on much larger caches), thus making its performance comparable with DoH.

Such competitive performance and strong privacy guarantees do not come for free. PDNS requires a significant effort to a ReR to handle queries fast. My benchmarking on an 8-core 3.0GHz AMD EPYC shows that a PDNS ReR can only handle few queries per second, while DoH can handle hundreds of queries per second. This implies a higher deployment cost for an operator, which can be absorbed via a subscription model for privacy-oriented customers, as currently done by Virtual Private Network providers. Indeed, my analysis concludes that a subscription fee of $5 per user is sufficient to make PDNS financially viable (see § 3.6.3). Further, participating ANSes need to support DoH, which causes a significant bandwidth increase. Nevertheless, such DoH adoption is not only meant for PDNS but also beneficial to current DNS, as it amends an existing user privacy violation [181, 255]. DoH for ANSes has been proposed independently from PDNS [188].

One final question remains: *what are the incentives for the adoption of PDNS?* For users, the extra privacy provided justifies the minor performance penalty. For the ReR, the extra cost is justified by unprecedented privacy guarantees, which could be offered at a premium. Participating

Table 3.1: Comparison of privacy-preserving properties of current DNS solutions versus single and multi-server PIR.

Solution	Defend Pervasive Monitoring	Hide Individual Access Pattern	Hide Organizational or Regional Access Pattern	Survive Non-Collusion Agreement Violation
DoUDP [49] / DoTCP [157]	No	No	No	N/A
DoT [193] / DoH [187]	Yes	No	No	N/A
DoT/DoH + Resolver Rotation [192, 267]	Yes	Yes*	No	N/A
ODNS [277] / ODoH [285]	Yes	Yes	No	No
ODNS/ODoH + Proxy Rotation [217]	Yes	Yes	Yes*	No
DoHoT [245, 246]	Yes	Yes	Yes*	Yes*
ReR-Less + DoUDP [278]	No	No	No	N/A
ReR-Less + DoH/DoT	Yes	Yes	Yes	N/A
DNS + Multi-Server PIR	Yes	Yes	Yes	No
DNS + Single-Server PIR	**Yes**	**Yes**	**Yes**	**Yes**

ANSes also have an incentive to support PDNS, as the additional traffic is offset by the increased privacy they can provide to their users, a valuable asset for competing domains especially when offering sensitive content.

3.1 Background and Motivation

In this section, I present the evolution of DNS along with its user privacy properties. Next, I offer some background on Private Information Retrieval (PIR) and analyze its potential privacy benefits when applied to DNS. I finally conclude the section discussing the key challenges when applying PIR to DNS.

3.1.1 DNS and Privacy

DNS clients send queries to a ReR, either run by an ISP or by public providers such as Google [82] and Cloudflare [79]. The ReR uses a cache to speed up DNS queries; cache misses trigger *iterative* DNS lookups to the ANSes for the root, top-level domain, and final zones ("root/TLD/final ANS" for short), before returning an answer to the user while updating the ReR's cache. The DNS RFC [49] specifies to send DNS queries either via UDP (DoUDP) or TCP (DoTCP). UDP was adopted in most cases because of better performance given due to its absence of connection handshake [211].

The original DNS protocol does not use encryption, potentially exposing user privacy to in-network eavesdroppers. DoT [193] and DoH [187] are two recent IETF standards which extend DNS by requiring the client to establish an encrypted session with the ReR. While DoT/DoH protects user privacy from in-network eavesdroppers, the ReRs still have full visibility of the DNS queries from their users. This represents a considerable privacy breach, especially in presence of public ReR with massive user bases like Google and Cloudflare.

Oblivious-DoH (ODoH) [208, 285] introduces an *oblivious* proxy between user and ReR. Assuming the ReR does not collude with the oblivious proxy, user identity and DNS queries are disjoint. However, non-collusion is hard to enforce and verify. Further, ODoH resolvers still learn the access pattern, *i.e.,* the frequencies of queried domains, for a particular organization or region when the DNS client defaults to the "closest" proxy and ODoH resolver, as commonly done for performance reasons [285]. To hide organizational/regional access patterns, DNS clients need to rotate their proxy and/or ODoH resolver frequently [192, 217, 267]. This causes performance degradation, *e.g.,* when a far proxy is selected, and a challenge to extend the non-collusion agreements among multiple ReRs and proxies. Further, simply rotating proxies or ReRs cannot protect individual user privacy in the long run; the best versions of this approach are capable of offering

K-anonymity [186, 191, 216], *i.e.,* preventing ReRs from differentiating between individual users, but not regional access patterns preservation.

In [278], the authors have recently proposed the provocative idea of eliminating all ReRs, thereby addressing DNS privacy issues associated with them. This approach has several concerning shortcomings. First and foremost, it introduces DNS "flattening"; DNS is hierarchical to provide *speed*, as a closeby resolver would respond if it can, *scalability*, as distributed caching avoids redundant queries, and *reliability*, as distributed caching allows to cope with failures at ANSes. As discussed in [278], such flattening would increase the overall DNS load by a few times, potentially becoming unsustainable for ANSes of popular or root zones [171]. Second, this approach is currently impractical as many ANSes, *e.g.,* Akamai [279], adopt complex rate limiting solutions to prevent traffic from non ReRs. Finally, ANSes currently do not support HTTPS (see § 3.3.3) thus requiring [278] to rely on DoUDP. This implies that pervasive traffic monitoring is possible under such approach, which indeed deteriorates the overall DNS privacy. A more robust solution should involve implementing DoH across all ANSes ("ReR-Less DoH"), which would further increase the overall DNS load.

3.1.2 Private Information Retrieval

PIR protocols [95, 141, 147, 148, 204, 209, 236] are advanced cryptographic techniques which allow a client to fetch an item from a remote database, *e.g.,* the ReR cache in the DNS scenario, without letting the server know which item it is. At a high level, PIR protocols are divided into *single-server* and *multi-server*, referring to how many server-side components they rely on. Overall, multi-server PIR provides efficient data transmission between the user and the servers, and lightweight computation, but it requires at least two non-colluding servers [148, 204, 209, 236]. It also requires intensive synchronization between servers for the maintenance of identical databases.

Single-server PIR protocols only require one untrusted server but rely on heavier cryptographic operations [95, 183, 241], which make them slower than multi-server PIR. Despite the latter, we argue that a robust privacy-preserving DNS should not rely on a non-collusion agreement, and thus discard multi-server PIR solutions.

PIR protocols can also be *stateful* or *stateless*. Stateful protocols [148, 183, 209] require the user to maintain a state, which contains information to generate PIR queries. The state is fetched from the PIR server and expires when the database is altered or the state is used a maximum number of times. With stateless protocols [95, 204, 241], the user does not store and update any database-related state except for the query keys. Given that a DNS cache changes frequently, stateless PIR should be preferred. I offer more details on single-server stateless PIR below.

Prior to delving into PIR, I introduce homomorphic encryption, which is the key component of PIR schemes within this category [95, 241, 247].

Homomorphic Encryption (HE) – It allows to perform computation on encrypted data [113, 173, 264]. HE relies on a cryptographic computational hard assumption known as learning-with-error (LWE) [235, 264]. Define an HE scheme $\mathcal{HE} : \{\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}\}$ which contains the following algorithms:

1. $\mathsf{KeyGen}(1^\kappa) \rightarrow (\mathsf{sk}, \mathsf{pk})$: Define a security parameter κ, 1^κ is a canonical notation to define the strength of an cryptographic scheme. This algorithm outputs a secret key sk and a public key pk.

2. $\mathsf{Enc}(\mathsf{sk}, x) \rightarrow [x]$: On input the secret key sk and an integer x, the encryption algorithm outputs a ciphertext $[x]$, which hides the x. I denote an integer in [] as an encrypted value.

3. $\mathsf{Dec}(\mathsf{sk}, [x]) \rightarrow x$: On input the secret key sk and a ciphertext $[x]$, the decryption algorithm outputs a plaintext x.

4. $\mathsf{Eval}(\mathsf{pk}, [x], [y], g) \rightarrow [z]$: On input the public key pk, ciphertexts $([x], [y])$, and an operator $g \in \{\mathsf{Add}, \mathsf{Mult}\}$, the evaluation algorithm outputs a ciphertext $[z] = g([x], [y])$. Depends on g, it is either $z = x + y$ or $z = x * y$.

For $\mathcal{HE}$ to be secure, it is essential that adversaries without access to the secret key are unable to obtain any information about the encrypted values. Homomorphism requires that for any operation $g \in (\mathsf{Add}, \mathsf{Mult})$ and a ciphertext $[z] = g([x], [y])$, anyone who holds a secret key can compute $\mathsf{Dec}(\mathsf{sk}, [z]) \rightarrow z$ which satisfies $z = g(x, y)$. The significant benefit of this property is that individuals with the public key can perform addition and multiplication operations directly on ciphertexts without requiring knowledge of the underlying values. Furthermore, it also supports a relaxed evaluation algorithm $\mathsf{Eval}(\mathsf{pk}, x, [y], g) \rightarrow [z]$ for relation $z = g(x, y)$. This implies the arithmetic operation between a plaintext and an encrypted value.

Construct PIR from HE – Suppose a PIR server maintains a cache $\mathcal{C} = (c_1, \ldots, c_N)$ and a user generates keys $(\mathsf{sk}, \mathsf{pk})$ by invoking KeyGen. The user shares pk with the server. To retrieve the i-th element c_i from the cache, the user encrypts a one-hot vector[1] $\vec{q} = (q_1, \ldots, q_N)$ in which only $q_i = 1$ but $q_j = 0$ for any $j \neq i$. An encrypted query $[\vec{q}] = ([q_1], \ldots, [q_N])$ is then transmitted to the server that performs the homomorphic evaluation by utilizing the algorithm Eval. Specifically, it computes the inner product $[r] = \mathcal{C} * [\vec{q}] = \sum_{j \in [N]} c_j \cdot [q_j]$ by repeatedly invoking Eval with operators Add and Mult, and then returns the response $[r]$ to the user. The user decrypts the response by $\mathsf{Dec}(\mathsf{sk}, [r]) \rightarrow r$. Due to the homomorphism, it satisfies that $r = \sum_{j \in [N]} c_j \cdot q_j = c_i \cdot q_i = c_i$.

Shrink Query – The concept described earlier enables the implementation of PIR at a cost of high communication overhead, as the entire encrypted query vector is transmitted. SealPIR [95] and

[1] A one-hot vector is a binary representation of a categorical variable in which only one element is set to 1 (hot) and the rest are set to 0 (cold).

Spiral [241] adopt different techniques for query compression and expansion which that shorten the query to a constant number of ciphertexts.

Specifically, define an algorithm $\mathsf{Expand}(\mathsf{pk}, [i]) \to [\vec{q}]$ which takes input an encrypted index $[i]$ and outputs a length-N query vector $\vec{q}$. The knowledge of sk is not needed to perform Expand, thus the server is able to construct $\vec{q}$ by itself given $[i]$. Based on this, the user only needs to send one ciphertext instead of N. I refer the readers to [95] (Section 3) and [241] (Section 2.1) for more details.

Optimize Query Processing – Note that the above approach is not feasible when N is large, *e.g.,* $N = 4,096$ [95]. The problem is overcome by representing $\mathcal{C}$ as a multi-dimension hypercube [95, 241]. Take the two-dimension case as an example. Define parameters m, ℓ such that $m\ell = N$. The server constructs its cache $\mathcal{C} = (\vec{c}_1, \ldots, \vec{c}_m)$ where each row contains $\vec{c}_i = (c_i^1, \ldots, c_i^\ell)$ for $i \in [m]$. To fetch c_i^j, the user constructs two ciphertexts $([i], [j])$. The server expands the queries to one-hot vectors $([\vec{q}_1], [\vec{q}_2])$, each having the i-th or j-th slot to be 1. The server first performs an HE evaluation on $[\mathcal{C}]$ and $[\vec{q}_1]$ to extract the row $[\vec{c}_i]$, then computes another inner product on $[\vec{c}_i]$ and $[\vec{q}_2]$ to extract $[c_i^j]$.

Observe that the first inner product only involves the multiplication between plaintext and ciphertext, while the second operates purely on ciphertexts. The HE algorithm adopted in SealPIR, called FV [167], has limited ability to perform the latter one, which results in slow query processing and large response size. Spiral proposes a combination of Regev [264] and GSW [173] schemes which provides efficient ciphertext-ciphertext multiplication thus achieving better performance on query processing.

3.1.3 Goals and Challenges

Comprehensive Privacy – Table 3.1 summarizes the privacy-preserving properties of state-of-the-art DNS solutions. No existing solution guarantees comprehensive privacy protections, *i.e.,* defending from collusion or passive data collection up to regional access pattern analysis, which is the main goal of this project. The hypothetical ReR-Less DoH can offer such comprehensive privacy, but it weakens the overall DNS performance, security, and reliability. Single-server PIR has the potential to offer the most enhanced privacy to DNS users as of today, but it comes with several extra challenges I discuss below.

Cache Population – PIR guarantees that the ReR cannot identify the queried domains. This also implies that PIR prevents a ReR from populating its cache, which invalidates its function. A strawman solution consists of bypassing PIR in presence of cache misses, *e.g.,* resorting to regular or ODoH. However, both solutions would relax the privacy constraints and put user privacy at risk. I propose a slight DNS modification wherein clients directly resolve DNS cache misses, and final ANSes populate a ReR's cache (see § 3.3.2).

Performance – The recent DNS evolution in the interest of user privacy has caused a slowdown in DNS queries. For instance, DoH requires at least three times the query time of DoUDP because of the handshakes to establish an encrypted channel. The handshake can be avoided if the HTTPS connection is re-used. However, this does not apply to DoH with proxy rotation which provides better privacy guarantees (see Table 3.1). In my measurements (see § 3.6.2), the median query duration for DoH is 69 ms, versus 25 ms for DoUDP, and it grows to about 272 ms for ODoH. Some previous studies [112, 285] report similar results while others [120, 139] report much higher values depending on the user location and distance to the ReR.

The introduction of PIR in DNS brings further slowdowns due to its additional complexity.

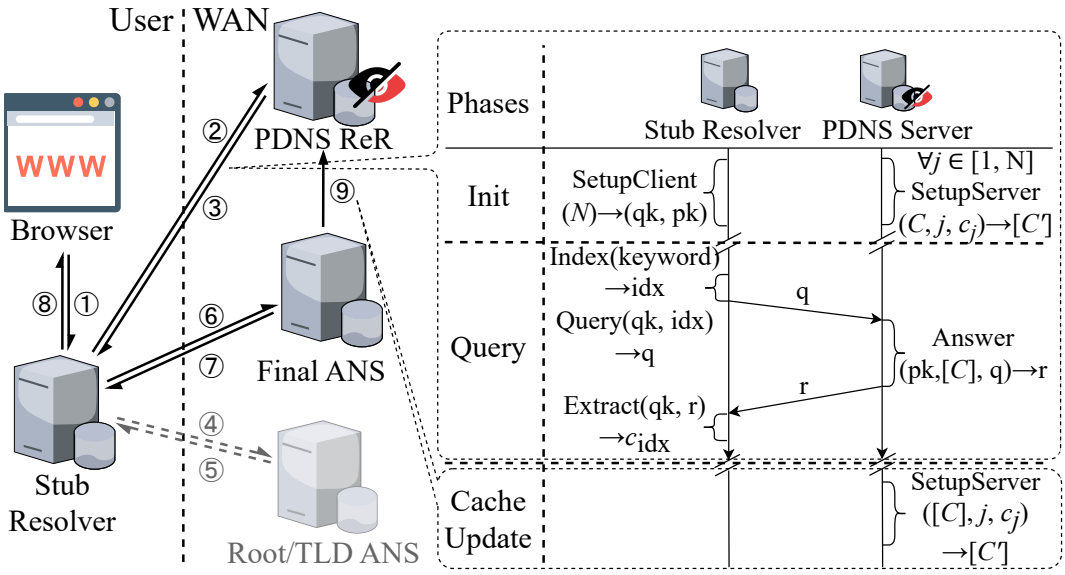


Figure 3.1: Visualization of PDNS and its workflow.

On the one hand, this is expected and understood by users as a trade-off for additional privacy, as commonly experienced in privacy tools like Tor [40] or VPNs [169]. On the other hand, a key challenge is to conceive a design that minimizes such overhead and achieves query times competitive with the state of the art. I plan to do so by carefully selecting the PIR scheme, and potentially modifying its implementation where performance bottlenecks are detected. Similarly, I will explore optimizations in DNS record storage and transmission.

Compatibility With Existing DNS – My goal is to enhance DNS with comprehensive privacy, rather than a complete overhaul. Specifically, I envision a design change similar to DoH, with minimal modifications required for clients and servers (ReR and ANS). While this constrains the design space, it also reduces the barriers to adoption.

3.2 PDNS Overview

This section overviews *PDNS* (a Privacy-Preserving DNS based on PIR), which augments DNS with PIR to provide query privacy against both eavesdroppers and snoopy ReRs. Figure 3.1 visualizes the workflow of PDNS both at a high level (on the left) and reporting its key PIR "primitives" (on the right).

PDNS replaces regular DNS queries with encrypted queries using a single-server PIR strategy. In presence of cache misses, the *stub resolver* (or DNS client, for simplicity) directly queries the authoritative name server of the domain to be resolved. The authoritative name server answers the client's query and forwards a *copy* of the returned DNS record also to the PDNS ReR, privately populating its cache. The authoritative name server learns the IP address of the PDNS ReR used by the client via a slight modification of edns-client-subnet extension of EDNS(0) [150] – a DNS extension allowing a ReR to share the client's IP address with an authoritative DNS.

In the remainder of this section, I define the privacy and threat models. Next, I define some fundamental PIR primitives which I use to formulate PDNS's workflow.

3.2.1 Privacy and Threat Models

Privacy Model – There are three main actors in the DNS ecosystem which can violate the user privacy: ReRs, ANSes, and any in-network device capable of intercepting DNS traffic. I assume *untrusted* ReRs which may track and inspect DNS queries, or deliberately drop DNS records from their cache – attempting to infer which user might query such domain when the record is populated into the cache again. I also assume that DNS traffic can be intercepted by third parties, *i.e.,* middleboxes interposing between DNS clients and both ReRs and ANSes. However, I assume that the attackers cannot break cryptographic primitives.

With respect to ANSes, their role requires some further discussion. ANSes operate at different levels in DNS, *e.g.,* from *non-final* ANSes which are responsible for large domains like `.com`, to *final* ANSes which are responsible for one or just a few domains. Queries to non-final ANSes are less sensitive as they reveal only partial information – under the assumption that DNS query minimization [109, 154, 237] is used, *e.g.,* avoid forwarding the full domain at each step. The privacy leak of a query increases as I approach the final ANS, since the full domain name is required in each query. However, it has to be noted that such ANSes are either operated by the same organization as the target domain, or by an organization contracted by the domain provider, *e.g.,* when leveraging Amazon Route53 [2]. It follows that such DNS queries do not leak any extra private information about a user than what the subsequent traffic directed to the domain, *e.g.,* HTTP(S) in case of a webpage. Unfortunately, protecting users from leaking privacy in IP/HTTP logs requires synergies from other privacy mechanisms for the Web [90, 152, 179, 182, 254, 315], and is beyond the scope of DNS and hence the scope of this chapter. Last but not least, ANSes are not in the position to gain access to a full individual or regional access pattern without a prohibitively high cost. In conclusion, I assume that ANSes cannot be *fully* trusted but they are not a critical DNS actor with respect to user privacy, differently from ReRs.

Threat Model – Many threats exist for DNS today, such as amplification, snooping, and flooding (or DoS) attacks [94, 101, 263, 279]. Overall, existing solutions to counter such attacks are still viable in PDNS. However, PDNS departs from the regular DNS workflow requiring its users to bypass the ReR and directly perform iterative DNS lookups (steps ④ – ⑦ in Figure 3.1) in presence of cache misses. While this is not a threat per se, it invalidates a common practice adopted by large ANS providers like Akamai – serving millions of queries per second [279] – which limit requests from non-well-known ReRs to protect against potential DoS attacks. Such rate limitation works in the current DNS where users are supposed to perform their queries recursively

(see § 3.1.1), but would fail in PDNS when handling cache misses. Note that this applies even more to solutions like ReR-Less [278], which fully bypass ReRs.

Finally, ANSes in PDNS are tasked to populate a ReR's cache, and could be misused to DoS a ReR via "reflection". This attack can be performed both by a malicious ANS or by an attacker disguising as an ANS. § 3.4 presents a defense mechanism to handle both reflection and DoS attacks. This mechanism relies on validating PDNS cache misses, and can thus not be applied to solutions like ReR-Less [278].

3.2.2 PIR Primitives

PIR schemes assume a key-value database $\mathcal{C} = (c_1, \ldots, c_N)$ of size N where the i-th key-value pair is defined as (i, c_i). All entries c_i are of the same length. I here define several PIR *primitives* which are the founding blocks of most single-server stateless PIR schemes. Assuming that the database of size N is known to both user and server, they first execute a one-time setup for the system.

- $\mathsf{SetupServer}(\mathcal{C}, N) \rightarrow ([\mathcal{C}])$: Given as input the database and its size, the server executes the *SetupServer* primitive which outputs an encoded database $[\mathcal{C}]$. The way a server encodes the plaintext database is specific to the PIR scheme. Note that PIR schemes generally do not support dynamic databases where entries can be modified after setup. However, most single-server stateless PIR schemes [95, 241] allow the server to re-encode the updated entry without repeating the whole SetupServer procedure.

- $\mathsf{SetupUser}(N) \rightarrow (\mathsf{qk}, \mathsf{pk})$: Given as input the database size, the user executes the *SetupUser* primitive which outputs a query key qk and a public key pk. The user stores qk as a private key and sends pk to the server. This step is only needed the first time a user connects to a

server, or when the user generates a new pair of keys. § 3.8 discusses how to share pk in a real world deployment consisting of multiple clusters of ReRs.

After setup, the server can answer PIR queries from the user. The following primitives are associated with PIR queries.

- $\mathsf{Index}(\mathsf{keyword}) \rightarrow \mathsf{idx}$: Given as input a keyword of the target record, *i.e.,* the domain name in the DNS scenario, the user or server executes the *Index* primitive, which outputs an index such that $1 \leq \mathsf{idx} \leq N$.

- $\mathsf{Query}(\mathsf{qk}, \mathsf{idx}) \rightarrow \mathsf{q}$: Given as input a query key and an index, the user executes the *Query* primitive which outputs a query q. It is a ciphertext that encrypts idx.

- $\mathsf{Answer}(\mathsf{pk}, [\mathcal{C}], \mathsf{q}) \rightarrow \mathsf{r}$: Given as input the public key of the user, the encoded database, and a query, the server executes the *Answer* primitive, which outputs a response r. The response is a ciphertext encrypting a message that contains c_{idx}.

- $\mathsf{Extract}(\mathsf{qk}, \mathsf{r}) \rightarrow c_{\mathsf{idx}}$: Given as input the query key and a response, the user executes the *Extract* primitive which outputs the idx-th database record c_{idx}.

3.2.3 Workflow

PDNS workflow consists of three main parts: initialization, query, and cache update (see Figure 3.1).

Initialization – Given as input X_0 initial DNS records, the PDNS ReR constructs a PIR DNS cache $\mathcal{C} := (c_1, \ldots, c_N)$ and executes the $\mathsf{SetupServer}$ primitive to obtain an encoded cache $[\mathcal{C}]$. Upon registering to a PDNS ReR, the user executes $\mathsf{SetupUser}(N) \rightarrow (\mathsf{qk}, \mathsf{pk})$ to derive the query key qk and public key pk. The user sends pk to the ReR, which needs it to answer private DNS

queries. This is a per-user key that can be shared across multiple ReRs, *e.g.,* in the case of a cloud DNS with multiple machines for load balancing (see § 3.8).

Query – A DNS query in PDNS implies the following steps:

1. A user who wants to visit a domain d executes $\mathsf{Index}(\mathsf{d}) \rightarrow \mathsf{idx}$. idx is a hash result of the domain d and it points to a specific slot in ReR's cache, where the DNS record for d might be located. User executes $\mathsf{Query}(\mathsf{qk}, \mathsf{idx}) \rightarrow [\mathsf{q}]$ and sends the encrypted query $[\mathsf{q}]$ to the PDNS ReR (②).

2. The PDNS ReR executes $\mathsf{Answer}(\mathsf{pk}, [\mathcal{C}], [\mathsf{q}]) \rightarrow [\mathsf{r}]$. The output $[\mathsf{r}]$ is a ciphertext that encrypts the corresponding cache slot, and is kept secret from ReR. The ReR sends $[\mathsf{r}]$ to user (③).

3. User executes $\mathsf{Extract}(\mathsf{qk}, [\mathsf{r}]) \rightarrow c_{\mathsf{idx}}$. If c_{idx} contains a valid DNS record for the domain d, the DNS query is terminated. Otherwise, the user performs an iterative DNS lookup (④ – ⑦). Note that PDNS attempts to speed up such iterative DNS lookup by providing in c_{idx} the NS-record of d, or the IP address of the ANS for d (thereby skipping ④ and ⑤, see § 3.3.2). The ANS could optionally invoke an authenticity request which asks the user to prove the existence of cache miss (see § 3.4).

Cache Update – The cache update happens after a cache miss is triggered and the user finishes an iterative DNS lookup for a domain d. The final ANS for d populates the PDNS ReR's cache by sending its most recent DNS record for d (⑨). PDNS ReR constructs a new entry c_j that contains the new record, and executes $\mathsf{SetupServer}([\mathcal{C}], j, c_j) \rightarrow ([\mathcal{C}'])$ to obtain a new encoded cache. As long as the final ANS does not "collude" with the ReR, the user's privacy is maintained. This is not a violation of my privacy model; in fact, rather than being a case of collusion, it is more accurately

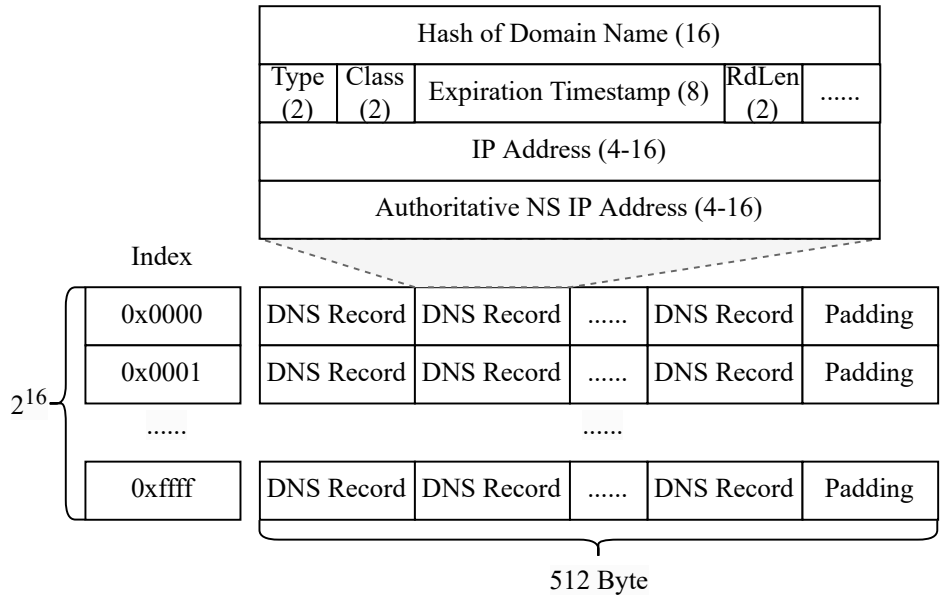


Figure 3.2: PDNS cache construction as a hash table with 2^{16} slots (*i.e.,* index range from 0x0000 to 0xffff). Each slot holds colliding DNS records in a priority queue ordered by expiration time. The size of a record is 38 Bytes (IPv4) or 62 Bytes (IPv6), assuming one IP per domain and its authoritative name server (NS).

described as a unidirectional information transfer from the final ANS to the ReR, as discussed in § 3.2.1.

3.3 PDNS Deep Dive

This section dives into the design details of PDNS.

3.3.1 DNS Cache Construction

PIR schemes assume a key-value cache $\mathcal{C} = (c_1, \ldots, c_N)$, *i.e.,* a hash table where a standard hash function H – known by both server and users – realizes $\mathsf{Index}(\cdot)$ (see § 3.2.2). A hash table is similar to the data structure used by current ReRs for their cache, *e.g.,* the popular BIND9 [3]. Yet,

one critical difference is that, with PIR, the capacity of the hash table is determined beforehand and all slots – no matter if occupied by DNS records or placeholders – have to be encoded into $[\mathcal{C}]$ via the SetupServer primitive (see § 3.2.2). At run time, DNS records are inserted by re-encoding a placeholder content at the slot indicated by the hash function.

Generally speaking, the query time of PIR increases as the capacity of the hash table increases. There is thus an incentive on reducing its size which in turn triggers more hash *collisions*, *i.e.,* multiple entries hashing to the same slot. Previous PIR schemes use Cuckoo hashing – where colliding entries are hashed with a second function – to minimize the hash collision rate [92, 248]. The drawback of Cuckoo hashing is that the users need to send multiple queries, one per hash function used, which can increase the query time by at least $2x$.

Recent PIR schemes such as Spiral [241] offer faster query time in presence of larger cache slots. Accordingly, instead of reducing hash collisions via Cuckoo hashing, I leverage hash collisions to purposely build large cache slots, which reduces the query time at the expense of more data to be returned to the user since the entire slot is returned (even if only containing placeholder data and no actual DNS records). To do so, I adopt a slight modification of *chaining* [243] (see Figure 3.2), where colliding entries in a slot are stored in a priority queue instead of the classic linked list. I order each priority queue using DNS record expiration times such that, once the queue overflows, the record that is most likely to expire is evicted.

3.3.2 Handling Cache Misses

In presence of cache misses, the user performs the iterative DNS lookup – although minimizing traffic to non-final ANSes as discussed next. After answering the DNS query from the user, the final ANS forwards the response to the PDNS ReR (whose IP was provided by the user) to populate its cache without leaking the IP address of the requesting user. This traffic is randomly delayed to

avoid the ReR correlating a previous query with a record update. Note that the final ANS might return a different response to the ReR occasionally when the client and ReR are geographically distant.

Minimizing Iterative Traffic – To shortcut the iterative lookup, and mitigate the privacy leak to root and TLD ANSes, I merge NS and A/AAAA records by appending the IP address of the final ANS at the end of the A/AAAA record (see Figure 3.2). The main drawback of this approach is that it consumes precious cache space (*e.g.,* 4 extra bytes per IPv4 address of the ANS added) which negatively affects the query time (see § 3.5.2). To regain some cache space, I replace the domain name field (variable length of maximum 256 bytes) with a hash value of the domain name – a fixed length of 16 bytes digest (from SHA-1 [103]). For simplicity, I only consider A/AAAA/NS DNS records in this project. However, PDNS is compatible with other types of DNS records since they are essentially strings with different lengths of up to 512B, which can be easily fit in large slots (tens of KBs, see § 3.5.2) of PDNS cache.

It is noteworthy that the above shortcut only applies to expired cached domains and will not work with uncached domains, which instead require full iterative DNS lookup at the client. I adopt this approach in PDNS because DNS record expiration is the main cause of the cache miss [119].

To identify cache misses at the client, another modification of the DNS record is required. Currently, an A/AAAA record contains: domain name, type, class, time-to-live (TTL), rdlength, and rdata [49]. The TTL field indicates for how long the received DNS record is valid. DNS records stored at the ReR also contain a timestamp of when the record was resolved. Each time a DNS query is matched in the cache, the ReR checks whether this record is expired. To allow the client to perform this operation, I replace the TTL field with a timestamp indicating when the record expires, *i.e.,* timestamp at insertion plus TTL, as illustrated in Figure 3.2.

The client communicates the IP address of its PDNS ReR to the ANS using a solution à la

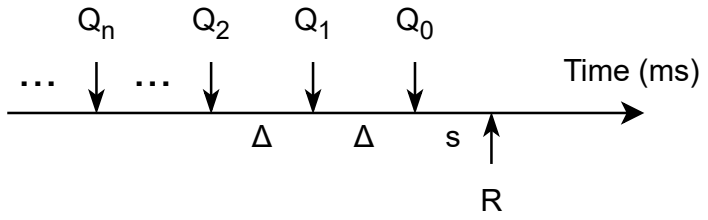


Figure 3.3: Illustration of the modeling for delayed response forwarding.

EDNS-PR, my own extension of EDNS(0) [150] similar to EDNS Client Subnet (ECS) [143] but with a different EDNS(0) `OPTION-CODE`. ECS is a DNS extension that allows a ReR to share the subnet of a client's IP address to an ANS for improved geolocation, enabling the ReR to resolve a domain to the closest available IP to the client. In PDNS, EDNS-PR includes the IP address of the ReR so that the ANS knows where to send the copied response to populate the ReR's cache.

Note that I use the full IP address for the PDNS ReR in EDNS-PR, instead of the subnet as suggested in ECS RFC [143]. The ECS RFC suggested using a subnet to protect the privacy of the user's IP address being communicated (and unfortunately mostly in plaintext with DoUDP today [49]), whereas in my proposal I send the IP address of the PDNS ReR (using encryption, see next section), which is not violating user's privacy.

Delayed Response Forwarding – PDNS ReR may perform a timing attack to correlate the user who finished the DNS query right before it receives the populated cache entry from the authoritative server. To mitigate this attack, the cache population traffic from the authoritative server to PDNS ReR should be randomly delayed.

Assume an infinite series of queries with descending subscripts $\ldots, Q_n, Q_{n-1}, \ldots, Q_1, Q_0$ arrive at the PDNS ReR, as shown in Figure 3.3. Each pair of adjacent queries is made by different users. I assume that these queries finish with an equal interval Δ, and each query has a probability of m to trigger the cache miss. Let Y_i be the random variable to denote whether a query Q_i has

triggered a cache miss. Then I have $\forall i \geq 0, \mathsf{Prob}(Y_i = 1) = m, \mathsf{Prob}(Y_i = 0) = 1 - m$. I further assume when a cache miss occurs, the response can be forwarded to PDNS ReR instantaneously. However, to defend against timing attacks from PDNS ReR, *i.e.,* correlating the forwarded response from ANSes and the queries where user identity is known to PDNS ReR, the ANS should delay the response forwarding by a random duration. Let X_i be the random variable of the delay of the response for query Q_i, sampled from distribution $\mathcal{D}$. For the sake of simplicity, I assume all ANSes have the same delay distribution.

Consider a response R arriving at time s after Q_0. The probability that this response is for Q_i is $\mathsf{Prob}(Y_i = 1 \cap X_i = s + i \cdot \Delta)$. Given that X_i and Y_i are two independent random variables, I have

$$\begin{aligned} & \mathsf{Prob}(Y_i = 1 \cap X_i = s + i \cdot \Delta) \\ = \ & \mathsf{Prob}(Y_i = 1) \cdot \mathsf{Prob}(X_i = s + i \cdot \Delta) \\ = \ & m \cdot \mathsf{Prob}(X = s + i \cdot \Delta). \end{aligned} \tag{3.1}$$

When $\mathsf{Prob}(X > \Delta) > 0$, meaning that the delay may be longer than Δ, there are multiple possible queries for which R might correlate to. It thus poses difficulty for the PDNS ReR to perform the timing attack. To quantify the difficulty, I use Shannon entropy as the metric. Specifically, the entropy of a response arriving s ms after Q_0 is the sum of the entropy of every past query

correlating to the response, *i.e.,*

$$\begin{aligned}\mathsf{Entropy}(s) = & \lim_{n\to\infty}\sum_{i=0}^{n} -\frac{\mathsf{Prob}(Y_i = 1 \cap X_i = s + i\cdot\Delta)}{\sum_{j=0}^{n}\mathsf{Prob}(Y_j = 1 \cap X_j = s + i\cdot\Delta)} \\ & \cdot log\frac{\mathsf{Prob}(Y_i = 1 \cap X_i = s + i\cdot\Delta)}{\sum_{j=0}^{n}\mathsf{Prob}(Y_j = 1 \cap X_j = s + i\cdot\Delta)} \\ = & \lim_{n\to\infty}\sum_{i=0}^{n} -\frac{\mathsf{Prob}(X_i = s + i\cdot\Delta)}{\sum_{j=0}^{n}\mathsf{Prob}(X_j = s + i\cdot\Delta)} \\ & \cdot log\frac{\mathsf{Prob}(X_i = s + i\cdot\Delta)}{\sum_{j=0}^{n}\mathsf{Prob}(X_j = s + i\cdot\Delta)}\end{aligned} \tag{3.2}$$

Equation 3.2 reveals a noteworthy point: the efficacy of the timing attack defense is not contingent upon the cache miss rate. This might seem counterintuitive. Nevertheless, the independence from the cache miss rate is rooted in the inherent inability of PDNS ReR – as guaranteed by the PIR – to determine whether a query will indeed result in a cache miss.

Gradual Deployment – PDNS can be gradually deployed using the following strategy. First, a PDNS ReR can still act as a regular resolver depending on the incoming queries, *e.g.,* queries not using PDNS encryption. Second, it can support PDNS for domains without participating ANSes by frequently querying such domains and building its local private DNS cache. This allows to quickly bootstrap adoption having the PDNS ReR provider bear some extra cost. Finally, ANSes supporting EDNS-PR will acknowledge this in their responses. This information can be used to inform the users, *e.g.,* via a browser icon similar to the security lock for HTTPS.

3.3.3 Communication Encryption

DoT or DoH are rarely supported by ANSes, *e.g.,* I tested 100 popular ANSes for different zones and found no support. Given that PDNS relies on iterative DNS lookups performed by its clients to handle cache misses (see § 3.3.2), the lack of encryption on this communication channel fails to

protect against in-network eavesdroppers, thus violating my privacy model (see § 3.2.1). It follows that ANSes participating in PDNS should adopt DoT/DoH to satisfy my privacy requirements.

On the path between user and PDNS ReR, PIR offers chosen-plaintext attack (CPA) security [205], meaning that, although the query content is kept secret from an attacker, *integrity* and *authentication* are not provided. It follows that an attacker can disrupt the service or even impersonate the ReR and attempt other attacks like phishing.

PDNS adopts HTTPS between client and ReR to guarantee integrity and authentication. I favor HTTPS over TLS since it makes it hard for third parties to distinguish between DNS and HTTP traffic given they both use port 443 [149]. It is also important to note that in proxy-based mechanisms (like ODoH), HTTPS connection reuse requires no correlation between client-proxy and proxy-ReR connections in order not to affect user privacy [285]. This constraint does not apply to PDNS as PDNS ReR is blind to DNS query contents.

3.4 System Security

Defend DoS and Reflection Attacks – PDNS's mechanism to handle cache misses is susceptible to (i) DoS on ANSes, and (ii) a "reflection" attack towards the ReR (see § 3.2.1). I here present a solution to protect against both attacks.

ANSes should only accept direct queries for a domain d from users who can *prove* an actual cache miss occurred at the ReR. In order to prove the existence of a cache miss, the user forwards to the ANS the encrypted query and its response (q, r) along with its secret key qk. To prevent a malicious user from counterfeiting (q, r), I require the PDNS ReR to sign an additional message containing the user IP, query timestamp, and the pair (q, r). With this information, the ANS can check whether q encrypts a query for d, and r indeed refers to a recent cache miss from the contacting IP.

The above solution has two major limitations though. First, sharing the secret key qk can be dangerous, as a misbehaving ANS could collude with PDNS ReR causing a privacy violation. Second, according to my performance evaluation (see § 3.5.2), this approach would bloat direct DNS queries with about 39KB extra traffic: 36.5KB from (q, r), about 2KB from the certificate, and few bytes for timestamp and IP address.

I solve both issues by asking for cache misses proofs only for *frequent* requests, *i.e.,* within a domain's TTL, which might indicate a potential attack. To do so, the ANS keeps track of the time at which it populated a domain record at a given PDNS ReR. Further, in presence of such (rare) proof request the user would send a new query for the same domain to the ReR using *backup* key pairs; this query is needed to generate a proof without leaking the user's main private key. After a successful proof, the user runs the *SetupUser* primitive ($<$0.15 sec, see § 3.5.2) to generate new backup key pairs. The ANS answers the direct DNS query right away regardless of whether the proof was requested, unless a pending proof for this IP already exists. However, it holds on the reception of the proof to update the PDNS ReR.

Validate ANSes – An attacker may impersonate an ANS and either DoS a PDNS ReR or pollute its cache. However, in PDNS, only ANSes should populate the cache of a ReR, which allows for simple access control. A PDNS ReR should only accept DNS records from IPs belonging to ANSes which can be verified via a regular DNS lookup or more strictly, a DNS lookup with DNSSEC [98] to avoid DNS hijacking [212], *i.e.,* confirming with top-level ANSes that the IP address of a sender matches that of the final ANS of the domain name contained in a DNS record.

Table 3.2: Summary and performance analysis of state-of-the-art single-server PIR solutions.

Solution	**When Cache Updates**	**Performance**	**# Number of Slots (S=64B)**			**Slot Size (NumSlots=2^{20})**		
			2^{16}	2^{18}	2^{20}	**128B**	**512B**	**2,048B**
SimplePIR [183]	Update required for server and every user	Update (MB/user)	7.4	14.7	29.5	42.4	86.8	178.1
		Query Duration (ms)	4.04	8.51	19.07	30.29	80.87	256.38
		Query Comm. (KB)	7	14	28	41	84	173
SealPIR [95]	Server update only	Update (μs/slot)	2.21	2.19	2.19	4.22	16.36	78.1
		Query Duration (ms)	117	301	902	1,636	5,831	25,338
		Query Comm. (KB)	278	278	278	278	278	278
Spiral [241]	Server update only	Update (μs/slot)	37.62	62.28	31.36	31.34	63.63	298.31
		Query Duration (ms)	249	501	794	797	1,423	3,882
		Query Comm. (KB)	30	30	36	36	36	36

3.5 Implementation

3.5.1 Implementation Details

PIR Scheme Selection – Table 3.2 lists three state-of-the-art single-server PIR solutions I have benchmarked, without optimization, for different cache and slot sizes over a single-core 3.0 GHz AMD EPYC CPU. SimplePIR [183] achieves, overall, the fastest query processing time but it is stateful, meaning that a user has to download a state (with size comparable to the square root of the database size) from the ReR whenever its DNS cache is updated. Given that a ReR's cache updates frequently, I discard this solution.

I instead favor single-server stateless PIRs: SealPIR [95] and Spiral [241]. SealPIR is slightly faster than Spiral when considering smaller cache and slot sizes, while Spiral outperforms SealPIR when assuming slots with large sizes. This is because Spiral does not provide optimal parameters for caches with small slot size; each slot with a size smaller than 256B simply gets padded to 256B. Spiral also requires less traffic than SealPIR (0.1x to 0.12x) for both the *Query* and *Answer* primitives. Though Spiral has a slightly longer update time than SealPIR, this is a server-only

operation requiring less than < 1 ms, and is thus not a decisive factor. Based on these observations, I select Spiral as the underlying PIR protocol for PDNS.

Spiral Code Optimizations – Table 3.2 shows that Spiral requires several hundreds or even thousands of milliseconds to complete a query, which is unacceptable for DNS. I here discuss two optimizations I have implemented to speed up Spiral.

First, Spiral's *Answer* primitive operates on 4 chunks of ciphertext for slots smaller than 2^{15} bytes; as the slot size increases, the number of chunks grows as a factor of 4. It follows that (at least) 4 concurrent threads can be used to parallelize and thus speed up the operations on each chunk. I leverage this observation to extend the current Spiral implementation to support increased multi-threading.

Second, Spiral's high CPU cost at the ReR is mostly due to the computation of the fast Fourier transformation, which can be optimized by pre-compiled single-instruction multiple-data operations that are already supported by x86 instruction set architecture. In my implementation, I enable Intel Advanced Vector Extensions 2 [15] which is already implemented by the Spiral library [5]. Such change shows a 2x speedup of the *Answer* primitive in my benchmark.

PDNS ReR and Client – I develop a custom PDNS ReR using the Rust-based Spiral PIR repository [5]. I opt for this approach, instead of extending an existing resolver, since PDNS ReR's workflow departs from a regular ReR, mainly due to the lack of iterative lookup to handle cache misses. I also leverage the Spiral repository to develop the PDNS client. While integrating the PDNS client into the OS would provide better performance, I opt for a DNS proxy for better flexibility and ease of adoption. The proxy intercepts outgoing DNS queries, transforms them into PIR queries, receives responses from PDNS ReR, and decodes the results. In presence of a cache miss, the proxy also performs (shortcut) iterative DNS lookups. In the end, the proxy constructs DNS responses itself using answers from either ReR or ANS, in presence of a cache miss, and returns

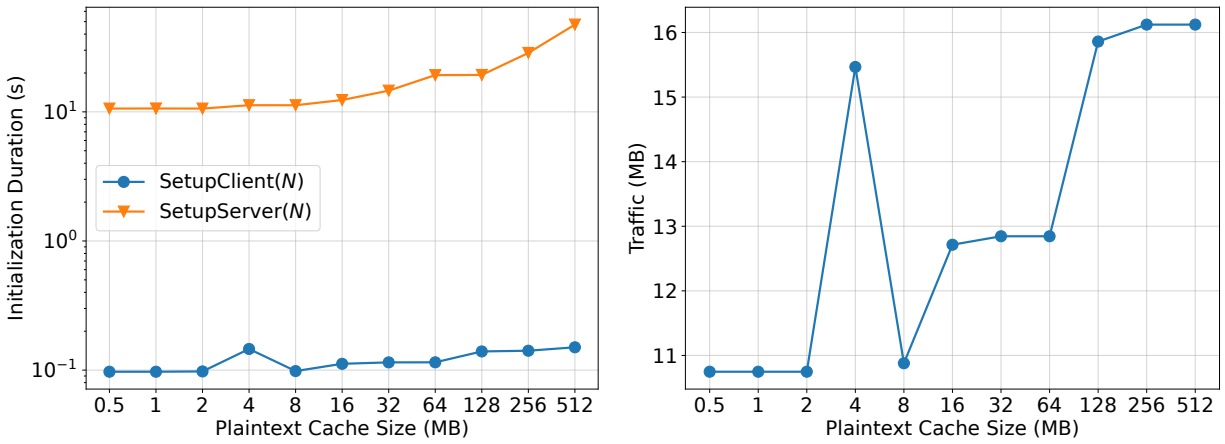
them to the OS.

Authoritative Name Server – PDNS requires three main changes at a participating ANS. First, support for EDNS-PR, my small extension of EDNS(0) to share the IP address of the ReR when a cache miss has occurred. Second, a routine to forward a DNS record to the ReR indicated in EDNS-PR. Third, a mechanism to challenge clients for proof of cache misses when needed (see § 3.4). I implement the above features as a patch (about 200 lines of code) of the popular Berkeley Internet Name Domain (BIND9) [3], a fully-fledged open-source resolver used by many ANSes worldwide [46].

3.5.2 Benchmarking

I benchmark all three components of PDNS: client, ReR, and ANS on machines equipped with an 8-core 3.0GHz AMD EPYC CPU and 8 GB RAM. This is an upgrade setup from [285] because PDNS is more computationally intensive compared to ODoH. I need at least 4 cores to speed up Spiral (see "Spiral Code Optimizations" in § 3.5.1) and at least 8GB memory to test a plaintext cache size of 512 MB (see "Query Resource Usage" below). Note that while PIR operations happen on the *encoded* cache, I use the size of the *plaintext* cache as a reference since it directly relates to the number of DNS records it can store. I ignore network latency in this benchmarking; I will instead introduce realistic latencies in the evaluation (§ 3.6).

System Initialization – As described in § 3.2.2, PDNS ReR and client execute SetupServer and SetupUser, respectively, at each reboot. Further, the client re-runs SetupUser when challenged by an ANS (see § 3.4) to regenerate its backup key pairs. Figure 3.4(a) shows the duration of each initialization phase (at client and server) as a function of the cache size. The figure shows that, with a cache size of 0.5MB, the server initialization takes 10 seconds, while it takes over 50 seconds with a cache size of 512MB. While long, this duration is acceptable since it is only required during



(a) Initialization duration as a function of cache size.

(b) Initialization communication cost as a function of cache size.

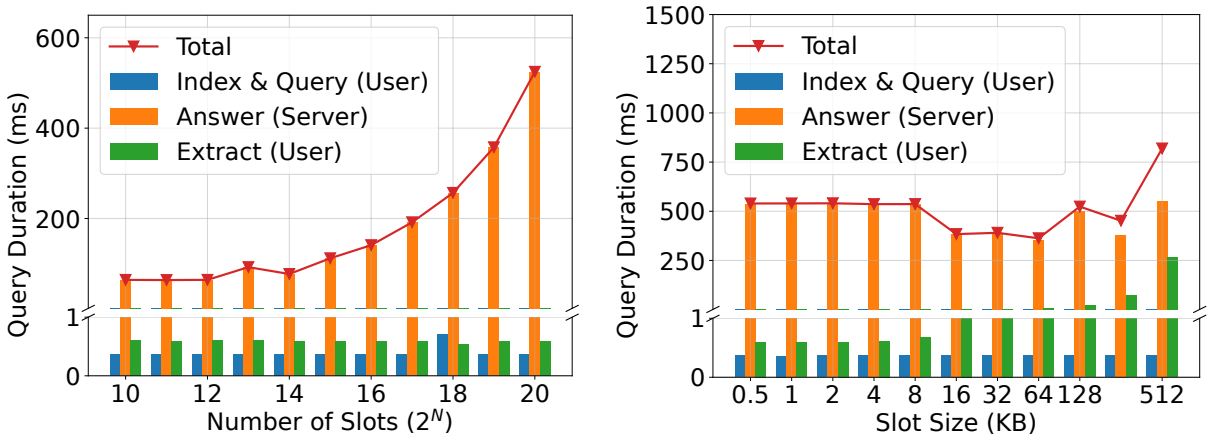
Figure 3.4: Benchmarking results for PDNS. Network latency is negligible.

a reboot of the ReR. Conversely, the client initialization is not a significant burden for the user as it only takes up to 150ms, irrespective of the cache size.

SetupUser also requires some traffic between PDNS client and ReR to share a public key pk. Figure 3.4(b) shows that this traffic increases as the cache size increases, *e.g.,* from 10.7MB with a small cache (0.5MB) up to 16MB when the cache size is larger than 128MB. An exception is observed when the cache size is 4MB where the traffic jumps to 15.5MB. This result is due to the selection of underlying cryptographic parameters by Spiral.

For the rest of the benchmarking, I assume PDNS was previously set up and ready to use, *i.e.,* I ignore the one-time cost of *SetupServer* and *SetupClient*.

Query Duration – Each PDNS query involves four PIR primitives: *Index*, *Query* and *Extract* at the user, and *Answer* at the PDNS ReR. Figure 3.5(a) shows the query duration for each primitive assuming (plaintext) caches composed of between 2^{10} (1K) and 2^{20} (1M) 512-Bytes slots, *i.e.,* total cache sizes ranging between 0.5 MB and 512 MB. These values are chosen because Spiral's authors selected optimal low-level cryptographic parameters for this range based on a heuristic



(a) Query duration as a function of number of slots with slot size of 512B.

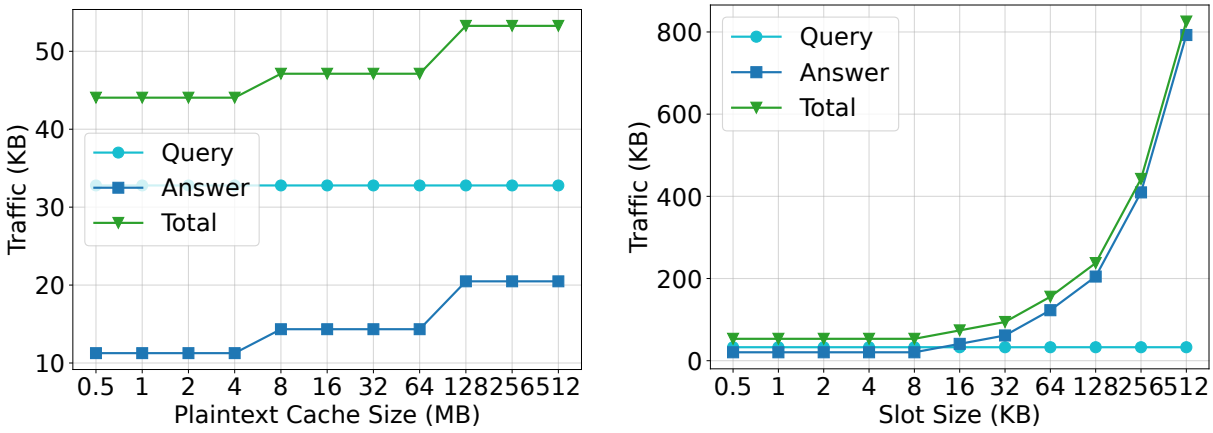
(b) Query duration as a function of slot size given a 512 MB cache.

Figure 3.5: More benchmarking results for PDNS. Network latency is negligible.

algorithm. It follows that each 512B-slot can hold about 13 DNS records with average size of 38 Bytes (see Figure 3.2). The PIR primitives at the user (*Index*, *Query*, and *Extract*) require sub-millisecond durations and, as expected, are independent from the cache size. The total query duration is dominated by the *Answer* PIR (at the server side) which requires between 80 ms – on a small cache with about 1,000 slots and up to about 13K DNS records – and up to 500 ms – on a large cache which might contain several million DNS records.

Next, I fix the cache size to 512 MB and investigate query duration as the number of slots increases, *i.e.,* the size of each slot decreases. For example, a 512 MB cache can be composed of either 2^{10} slots with a size $S = 512$KB, or 2^{20} slots with $S = 512$B. Figure 3.5(b) shows that the duration of the *Index* and *Query* primitives (at the user) are unaffected by the slot size. Conversely, the duration of the *Extract* primitive, also at the user, decreases as the slot size decreases, although non-linearly, *e.g.,* from 266 ms ($S = 512$KB) down to 21 ms ($S = 128$KB). This nonlinearity comes from the better optimizations of Spiral for large slot sizes.

For *Answer* (server side), the impact of the slot size is non-trivial due to the conflicts and



(a) Query and answer traffic as functions of cache size.

(b) Query and answer traffic as a function of slot size given a 512 MB cache.

Figure 3.6: More benchmarking results for PDNS.

compromises between multiple underlying cryptographic primitives. Its performance is best when $S = 64$KB and worsens when the slot size either increases or decreases. Given the *Answer* primitive has an overall much higher duration than all other primitives, the total duration of the PIR query is the fastest (362 ms) for $S = 64KB$ to which it corresponds 2^{13} slots in a large 512 MB cache. This is a $\sim$40% drop from 539 ms when assuming a small slot ($S = 512$B) and a $\sim$55% drop from 819 ms with a large slot ($S = 512$KB).

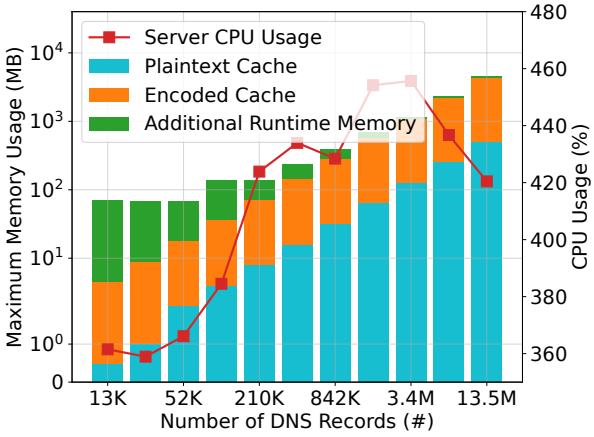
Query and Answer Traffic – First, I focus on the traffic between PDNS client and ReR. Figure 3.6(a) shows the traffic for both queries (PDNS client) and answers (ReR), assuming a slot size of 16KB and increasing cache sizes. I find that the query traffic is bound to 32KB since the query is the encryption of an index and hence independent from the cache size. The query and answer traffic only increases when the cache size grows from 4 to 8MB and from 64 to 128MB, but is constant otherwise. This is attributed to the selection of underlying cryptographic parameters by Spiral.

Figure 3.6(b) instead fixes the cache size to be 512 MB and shows traffic for both queries and

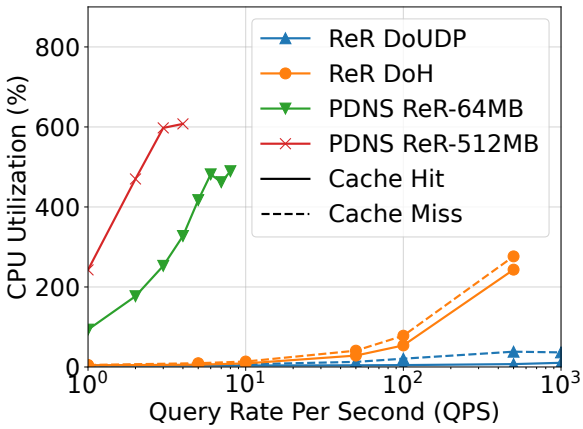
answers assuming increasing slot sizes (and hence decreasing numbers of slots). At the client, each query consumes a constant 32 KB independently of the slot size, again, due to that the query is the encryption of an index. At the ReR, the traffic increases as the slot size increases since a full slot is returned. For example, the answer to a query contains 20KB when there are 2^{16} slots ($S = 8KB$), versus nearly 800KB when there are 2^{10} slots ($S = 512KB$). Considering both query duration (Figure 3.5(b)) and traffic (Figure 3.6(b)), I select a cache *shape* for PDNS of 2^{15} slots with a size of 16KB, for a total cache size of 512MB.

Next, I focus on the traffic between PDNS client and an ANS in presence of cache misses. Since PDNS require DoH, this increases the traffic from around 200B, *i.e.,* a DoUDP query from ReR to an ANS, to about $\sim$7KB [112, 298] because of the TLS handshake – $\sim$1KB is needed instead if HTTPS connection is reused. Next, the ANS also needs to duplicate the query response to the PDNS ReR. In this case, there is no strict privacy requirement and thus DoUDP can be used ($\sim$100B). Overall, this is a significant bandwidth increase for an ANS mostly due to DoH. However, the current adoption of DoUDP between ReRs and ANSes is yet-another violation of user privacy when EDNS(0) is adopted [181, 255]. Proposals have been made to encrypt the channel between ReR and the ANS [188]. If this proposal is adopted, then PDNS only adds $\sim$100B, or less than 1.5% of extra traffic.

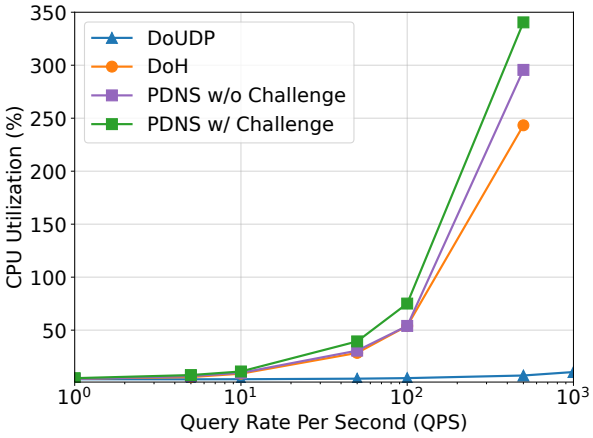
Resource Usage – Next, I benchmark CPU and RAM usage for PDNS. Figure 3.7(a) shows the RAM required by the plaintext cache, the encoded cache, and the runtime ReR, as a function of the (plaintext) cache size. The figure shows that the encoded cache requires 8x the memory used by the plaintext cache, and that runtime memory usage is comprised between 60 and 100MB. As a result, the memory usage of PDNS at a ReR ranges between 68MB and 4.3GB, assuming caches which can hold about 13K (512KB) and 13M (512MB) DNS records using IPv4 (*i.e.,* 38B as in



(a) Memory and CPU as a function of cache size.



(b) CPU usage of ReRs as a function of query rate.



(c) CPU usage of ANS as a function of the query rate.

Figure 3.7: More benchmarking results for PDNS. Network latency is negligible.

Figure 3.2). With respect to CPU usage[2], Figure 3.7(a) shows that it ranges between 360% and 460%. This is a result of the multi-threading implementation of the *Answer* primitive, and the usage of 4 concurrent threads (see § 3.5.1).

Scalability – PDNS intensive CPU usage is expected to limit its *query rate*, *i.e.,* the concurrent number of queries per second (QPS) it can handle. I compare the query rate of PDNS with both

[2]All numbers for CPU usage are concerning one core of a single CPU.

DoUDP and DoH provided by BIND9 [3]. I do not report numbers for ODoH since equivalent to DoH for the ReR. For each protocol, I increase the query rate until the query duration increases by 50%, and then report the previous CPU usage and rate. Figure 3.7(b) shows that DoUDP and DoH scale much better as they can serve roughly 1,000 QPS (CPU usage of 32%) and 500 QPS (CPU usage of 200%) before significantly delaying their respective query response times. In comparison, PDNS ReR reaches a query rate of 4 and 8 QPS for respectively a large (512 MB) and small cache (64 MB). While the ReR's CPU is not fully utilized (500-600%), adding even one extra query would increase the query duration by more than 50%.

Despite the current large performance discrepancy, the uprising specialized hardware would substantially improve the scalability of PDNS. For example, if the computation duration would decrease by 1,000x to within 1 ms [271], then PDNS would be able to reach at least 1,000 QPS without HTTPS or hundreds of QPS with HTTPS, similar to DoH in Figure 3.7(b). The figure also differentiates between queries triggering a cache hit or a miss. At their maximum query rate, the iterative process associated with a cache miss costs an extra 33% CPU usage for both DoUDP and DoH. No impact is observed for PDNS as cache misses are resolved at the client and not the ReR.

Finally, I also benchmark the CPU usage at the ANS, using the same methodology adopted for the ReR. Figure 3.7(c) shows that the CPU usage at ANS associated with DoUDP is negligible: only 7% with a rate of 1,000 QPS. DoH is instead much more challenging, but still reaches a rate of 500 QPS with a CPU usage of 240%. Note that the combination of extra bandwidth and CPU might explain the current lack of adoption of DoH among ANSes (see § 3.3.3), but is a requirement for PDNS. In addition to DoH, PDNS further adds 50% CPU usage to support EDNS-PR queries, *i.e.,* privately populate DNS records at a ReR, and another 50% to validate proof of cache miss for suspicious queries.

3.6 PDNS Evaluation

This section evaluates PDNS. Given that an actual PDNS deployment is challenging, *e.g.,* it requires participating ANSes and users, I resort to the next most realistic setup. I set up a test-bed consisting of a PDNS client, ReR, and a participating final ANS. Next, I simulate real network latencies and DNS queries I have collected in the wild. I then evaluate PDNS from four perspectives: query duration, privacy guarantees, resilience to attacks, and impact on Web performance. When possible, I compare PDNS performance against state-of-the-art DNS solutions: DoUDP, DoH, ODOH, and DoHoT. I further include in my evaluation a hypothetical ReR-Less DNS assuming both DoUDP and DoH. Before diving into the analysis of my evaluation, I first describe my test-bed setup and data collection in more details.

3.6.1 Methodology

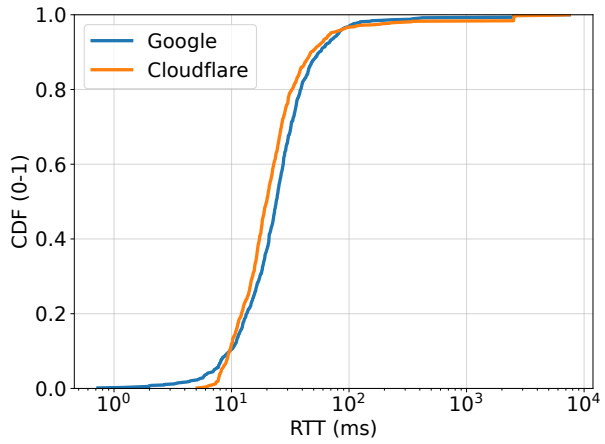
I evaluate existing DNS solutions (DoUDP, DoH, ODOH, and DoHoT) using experiments in the wild. I select Google DNS [82] and Cloudflare DNS [79] as target ReRs for both DoUDP and DoH. To emulate a realistic access network of a DNS client, I resort to Mysterium [21], the most popular DVPN (see chapter 2) which provided 1,415 Internet residential dVPN nodes from 62 countries at the time of conducting this measurement. To derive the duration of a DNS query between a Mysterium node and ReRs, I subtract the latency between my machine and the node for each RTT needed, *e.g.,* 1 RTT for DoUDP and 3 RTT for DoH given TLSv1.3 [266] and no connection reuse. I preferred Mysterium over academic platforms like the popular RIPE Atlas [287] since it offers higher flexibility, *e.g.,* allowing to send DoH and ODoH queries.

I use instead a single location (our lab) for DoHoT since I cannot force a Tor circuit between Mysterium nodes and a ReR. However, I restart Tor after each experiment which gives us 522

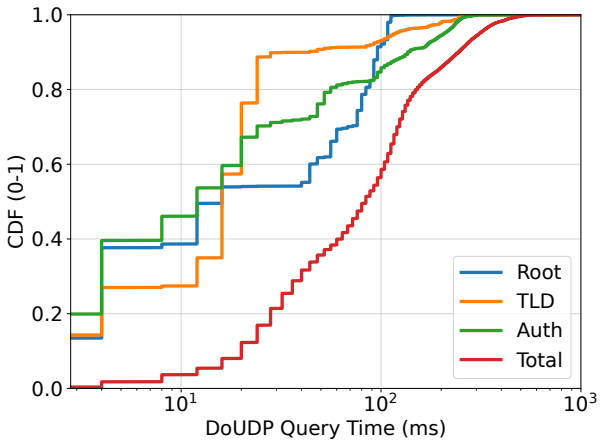
unique exit nodes over 24 hrs. For ODoH, I still rely on Mysterium but also iterate the three oblivious proxies provided by DNSCrypt Proxy [8], a popular and cross-platform local proxy which supports many DNS protocols. Finally, I perform iterative DNS lookups on my machine for 122K domains collected in Mysterium (see § 3.6.4), and estimate the duration of direct queries towards authoritative DNS servers, as used by ReR-Less DNS. This procedure was confined to my own machine due to the impracticality of executing all 122,000 iterative DNS lookups across each of the 1,415 dVPN nodes. It is crucial to mention that my machine, located in the United States, benefits from relatively high bandwidth and short latencies to ANSes. Additionally, my focus was exclusively on top-ranked sites. Consequently, my evaluation of the ReR-Less DNS performance should be viewed as a conservative estimation. Users with lower bandwidth and/or in developing countries might experience less favorable performance.

When experimenting with DNS in the wild, there is no control over whether queried domain names are cached or not at a ReR. To study the effect of cache *hit* or *miss* at a ReR, I query for my own domain names – registered at AWS Route 53 [2] – with TTLs of one second and one hour, respectively. The one-hour interval guarantees cache hits as long as my queries happen within such an interval. The one-second interval guarantees a cache miss as long as I perform queries slower than once a second. While many public ReRs would ignore such low TTL value, I have verified that Google and Cloudflare DNS both support it.

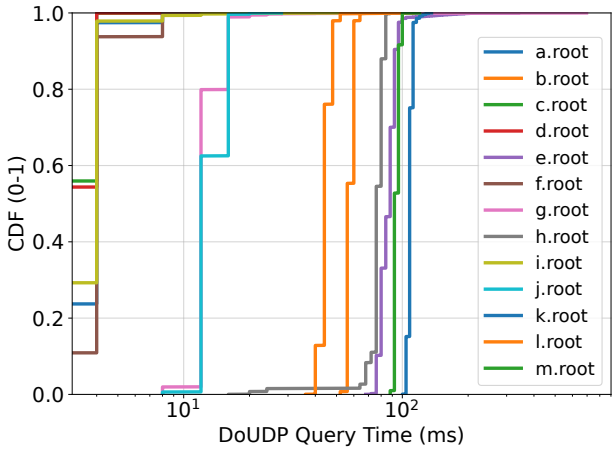
To evaluate PDNS, I instead set up a test-bed composed of PDNS client, ReR, and a participating final ANS. Each machine is equipped with the same hardware used in the benchmarking (see § 3.5.2). I then apply network delays between the machines using the Linux `Traffic Control` (tc) module [18] driven by the real latencies collected in the above experiments. I test a large (512MB) and small (64MB) cache using the best performing *shape* as from my benchmarking experiments, *e.g.,* 2^{15} slots with size $S = 16KB$ for the 512MB cache.



(a) CDF of RTT between 1,415 residential dVPN nodes and Google and Cloudflare DNS ReRs.



(b) CDF of the duration of iterative DNS queries distinguishing between root, TLD, and fianl ANSes.



(c) CDF of the duration of DoUDP queries per root ANS.

Figure 3.8: Analysis of active DNS measurements.

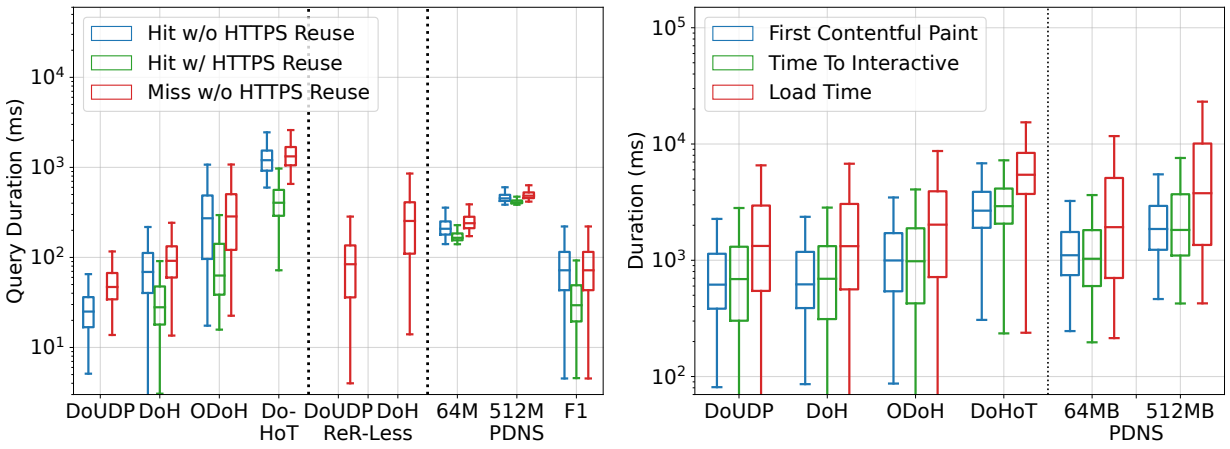
Ethical Considerations. My methodology uses synthetic DNS traffic via a publicly available service (Mysterium), avoiding ethical concerns. However, in § 3.6.4 I rely on DoUDP traces collected with a passive Mysterium node. To protect the privacy of Mysterium users, I discard user-related information such as their IP addresses. IRB at Northwestern University has deemed this data collection as non-human research, as I only collect non-identifiable private information without any accompanying data that could reveal individuals' identities.

DNS Measurement Result. I here present high-level results from my DNS measurement study. I first analyze the network delay towards popular ReRs. To do so, I connect to 1,415 residential dVPN (Mysterium) nodes and send `ping` (ICMP packets) to both Google and Cloudflare public DNS ReRs, measuring the round trip time (RTT) of the path <client, dVPN node, ReR >. Then, I derive the latency of the path <dVPN node, ReR > by subtracting the latency from the path <client, dVPN node> which I also obtain via `ping`. Figure 3.8(a) shows the CDF of the RTTs between 1,415 residential dVPN nodes and Google and Cloudflare DNS ReRs. Overall, faster RTTs are measured for Cloudflare, *e.g.,* a median RTT of 19.9ms compared to Google's median RTT of 24.0ms.

Next, I analyze the iterative DNS lookups (DoUDP) I performed for 122K domains (40K SLDs) collected with a passive Mysterium node. Figure 3.8(b) shows the CDF of DoUDP query duration per domain distinguishing between each step of the iterative lookup: root, TLD, and final ANSes. The figure shows similar results across ANSes, with a median duration of 16ms for both root and TLD ANSes, and 12ms for final ANSes. Further analysis shows that the highly variable query duration observed for both root and TLD ANSes is primarily due to `dig` [4, 7] which iterates through different root and TLD ANSes. In fact, the closest root ANSes to my test-bed (c.root and d.root) consistently respond in less than 1ms, versus over 100ms for the furthest one (k.root) as shown in Figure 3.8(c). Similar results apply to TLD.

3.6.2 Performance

Query Performance. Figure 3.9(a) shows boxplots of query duration per DNS protocol. For PDNS, I differentiate between small (64MB, up to 1.6M entries) and large (512MB, up to 13M entries) caches, as well as a *futuristic* implementation relying on F1 hardware [271] with large cache, which provides *at least* 1,000x speedups to CPU. While I cannot obtain such hardware yet,



(a) Boxplots of query duration of different DNS protocols.

(b) Boxplots of Web performance of different DNS protocols.

Figure 3.9: Performance evaluation of PDNS.

I simulate its performance assuming that the Answer primitive would only require 1 ms. I further distinguish between queries which triggered cache misses or not, and whether HTTPS connection reuse was adopted, when applicable.

Figure 3.9(a) shows that DoUDP, *i.e.,* DNS without any security or privacy guarantees, achieves the best performance with a median query time of 25 ms, which is slightly longer than the median RTT of 22 ms between my emulated users (via Mysterium as detailed in § 3.6.1) and the ReR– thus confirming negligible time spent at both client and ReR to handle a DNS query. DoH almost triples the query duration from DoUDP (median of 69 ms) due to the two additional RTTs ($\sim$ 45ms) required by TLSv1.3; as expected, the query duration in DoH drops to 28 ms in presence of connection reuse. ODoH – which further enhances user privacy while sacrificing performance – achieves a median query duration of 272 ms, which reduces to 63 ms with connection reuse.

In the case of ReR-Less DNS, when utilizing DoUDP, the median latency is around 84 ms with considerable variability. When emulating[3] ReR-Less DoH, which is necessary to protect from

[3]Emulation is required since ANSes currently do not support DoH.

pervasive DNS monitoring, latency significantly increases, with median exceeding 250 ms and the third-quarter latency reaching above 450 ms. My analysis is based on the assumption of no HTTPS connection re-use, as ReR-Less DoH necessitates establishing new HTTPS connections with various ANSes, reducing the likelihood of benefiting from persistent HTTPS connections, except perhaps with root/TLD ANSes.

It is noteworthy that these results represent the performance of ReR-Less DoH from the perspective of a single user with relatively good bandwidth and location, and are limited to top-ranked domains. The performance for average users worldwide is likely to be substantially lower than what I have observed, due to an inflated network path to an ANS. Further, these results do not take into account of the extra load incurred on ANSes if ReRs are bypassed globally. Such loads can only be amortized via more hardware or by slowing down user queries.

PDNS achieves median query durations of 208 ms and 450 ms for cache sizes of 64MB and 512MB, respectively. This means that PDNS is slightly faster than DoH and ReR-Less DoH assuming a small cache, and no connection reuse. While HTTPS connection reuse is also beneficial to PDNS, the benefit is less evident compared to DoH and ODoH because CPU processing is the main bottleneck of PDNS. When considering a much bigger cache, PDNS pay a penalty of 180ms when compared with ODoH. Still, the median query latency is 2x faster than DoHoT. This shows that the performance of PDNS is already acceptable today, especially to the many people who care about their privacy.

It is important to highlight that the prospects for improving ReR-Less DNS or DoHoT are limited, as their performance is primarily constrained by network latencies, leaving little space for optimization. In contrast, there is significant potential for enhancing the performance of PDNS, as its latency is primarily due to computational delays. For example, when combining PDNS with specialized hardware (*e.g.,* the F1 accelerator [271]), its performance are comparable to DoH and

better than ODoH, while offering much stronger privacy guarantees (see Table 3.2). PDNS with F1 will also outperform ReR-Less DoUDP. Finally, the cache misses add ∼20ms to the query time for all DNS protocols except the ReR-Less DNS.

Web Performance. I use WebPageTest [44] to automate Chrome (on Linux) and both perform web page loads and collect telemetry, *e.g.,* classic web performance metrics: FirstContentful Paint (FCP), Time To Interactive (TTI), and PageLoadTime (PLT). I target the top 1,000 websites from Tranco [41] which are loaded 5 times per configuration (and then report the median of each metric). I test each website using all DNS solutions studied so far except ReR-Less DNS since no DoH is supported by ANSes today, and assuming HTTPS connection reuse, since it is a realistic behavior of different OSes and browsers [120]. I use Cloudflare as a public ReR for existing DNS protocols, and my ReR for PDNS while injecting the latency measured between testing client and Cloudflare (2-3ms). Note that my testing machine connects to the Internet with a symmetric upload/download bandwidth of about 100 Mbps.

Overall, the Web performance tests (Figure 3.9(b)) confirm the results of the query duration (Figure 3.9(a)), with DoUDP being the fastest protocol and DoHoT being the slowest. When assuming a small cache, PDNS has performance comparable with ODoH, within 100 ms per metric which can hardly be perceived by the user. Still, ODoH outperforms PDNS when using a large cache, saving ∼900 ms for FCP/TTI which would indeed be perceived by the user.

3.6.3 Deployment Cost Analysis

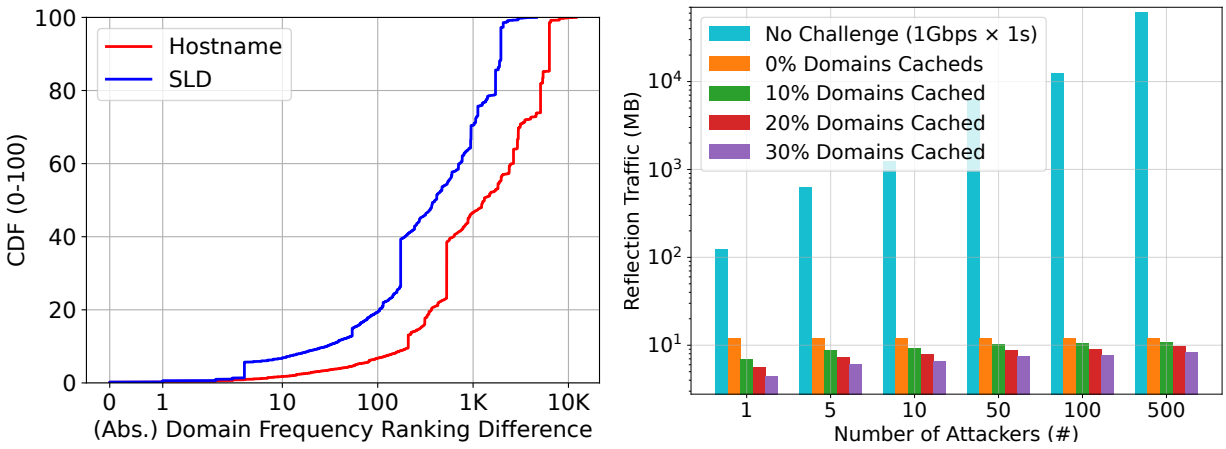
I here estimate the deployment costs of PDNS and discuss whether it is viable to be provided as a subscription-based service as of *today*. My analysis is based upon a comprehensive study [280] that models the DNS client behavior with a dataset collected from a university campus network.

I assume that PDNS is deployed using a public cloud service. Upon consulting the pricing

calculators provided by AWS [1] and Google Cloud [10], I determine that the costs associated with configuring an 8-core CPU machine – similar to the one used in the benchmarking (§ 3.5.2) and evaluation (§ 3.6) sections – are comparable across two cloud providers. Specifically, the estimated computing costs amount to approximately $148.92 on AWS and $148.37 on Google Cloud, per month. For simplicity, I will consider the rounded value of $149 for the subsequent calculations. It is worth noting that both costs scale linearly with the number of CPU cores. As shown in Figure 3.7(b)), PDNS running on a machine equipped with an 8-core CPU can handle 8 QPS (assuming a small cache) or 4 QPS (assuming a large cache). In addition, the cloud also charges for data transfer from cloud machines to the Internet (the reverse is free). AWS and Google Cloud charge for at most $0.09 per GB and $0.085 per GB, respectively. I take the upper bound of $0.09 per GB for the subsequent calculations.

According to the findings in [280], users perform on average between 2,600 and 3,724 DNS queries every day. It follows that a PDNS using the configuration above can serve at least between 93 (large cache) and 186 users (small cache) in a day, *i.e.,* $3600 \times 24 \times 8/3,724 = 186$. As a result, the computing cost per user amounts to approximately $0.8 with a small cache or $1.6 with a large cache. In addition, the data transfer will cost $3,724 \times 30 \times 40\text{KB} \times \$0.09/\text{GB} = \$0.4$ per user. This means that if a user is willing to pay more than $2 per month to safeguard their privacy, PDNS could be a viable business proposition.

The above analysis ignores potentially concurrent users as well as the bursty nature of DNS requests. In the study by [280], the bursty behavior of DNS queries is measured using “clusters”, where each cluster consists of a minimum number of 3 queries and is separated from other clusters by an idle period of 2.5 seconds. In their research, they report the CDFs of the number of queries per cluster and cluster duration. Although direct ratios between the number of queries and duration within each cluster were not reported, I sample these values at various percentiles and approximate



(a) CDF of domain frequency rank differences.

(b) Reflection traffic as a function of attackers.

Figure 3.10: Privacy and security evaluation of PDNS.

the bursty demand by calculating the ratios ourselves. Based on my analysis, I find that the bursty queries generated by up to 1,033 users reach a maximum of 126 QPS. Another study has reported similar peak QPS with many more users [171]. To support such capacity, machine(s) with 128 CPU cores are required with a small cache, or 256 CPU cores are needed with a large cache. Consequently, the monthly computing costs to serve one user would amount to approximately $128/8 \times 149/1,033 = \$2.3$ with a small cache, or \$4.6 with a large cache. Besides the data transfer cost of \$0.4 per user, I conclude that offering PDNS as a service would be financially viable if the monthly subscription fee is set at \$5 or higher.

3.6.4 Privacy and Security

Regional Access Pattern – § 3.1 concludes that PDNS provides the highest privacy guarantees. In particular, it does not require a non-collusion agreement and can hide regional access patterns from the ReR. The former is guaranteed by its cryptographic theorems [173, 264], while the latter is not obvious given that a PDNS ReR can still learn some information from the analysis of cache

misses.

Figure 3.10(a) shows the CDF of domain (hostname or SLD) popularity rank difference (absolute value) obtained with and without PIR (ground truth). For instance, "cnn.com" has a ranking difference of 23,270 since it is ranked 104,614th without PDNS but 81,344th assuming PDNS. This figure relies on a trace of 20 million DNS queries of 122K unique domains I collected. Overall, the figure confirms the privacy protection offered by PDNS as the overall ranking has been disrupted. For example, 99% of the 122K hostnames in my traces have an absolute ranking difference larger than 100 (and 40% larger than 10K). Similar results apply to SLD.

Information Exposure to ANSes – In addition to protecting user privacy from ReRs, PDNS also utilizes the presence of ReRs to decrease the frequency of iterative DNS lookups, thereby reducing information exposure to ANSes. In my simulation, I observed a cache miss rate of less than 10%. Within this rate, 90% was caused by expired entries, and 10% by uncached entries. While my cache miss rate is lower than some previous studies (30% was reported [119]), the pattern of cache misses being primarily due to expired entries is consistent with these findings. Therefore, in 90% of cache miss cases in PDNS, users only need to contact the final ANSes directly, significantly limiting exposure to non-final ANSes.

In short, PDNS cuts down 70 to 90% of the ANS queries required by ReR-Less DNS. For the remaining traffic directed to ANSes, only roughly 10% require interactions with non-final ANSes. Such limited exposure will likely disrupt the frequencies of sub-domains observed at non-final ANSes, similar to Figure 3.10(a). This achievement aligns with my goal to minimize user interactions with ANSes and reduce privacy risks, even though I consider these risks acceptable today. In contrast, ReR-Less DNS leads to the exposure of all user information to ANSes and, on average, causes a 4 to 16 times increase in their workload compared to the current norms [171]. The impact is even more pronounced at ANSes for popular domains, where the workload can be

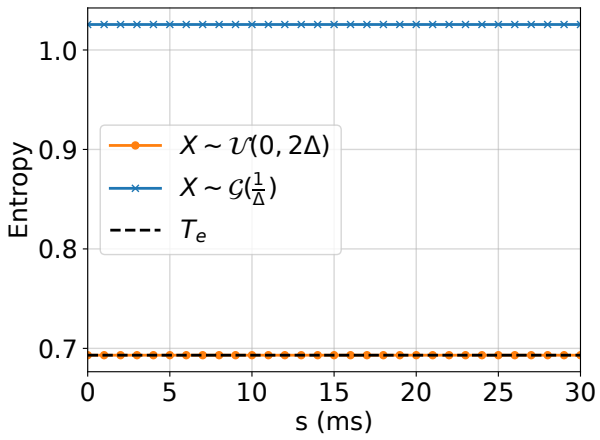
hundreds of times greater. Moreover, ReR-Less DNS introduces security vulnerabilities to ANSes.

Resilience to Reflection Attacks – Next, I evaluate PDNS resilience to reflection attacks. I use my DNS traces, including 42K ANSes involved and 122K unique domains, to drive the distribution of the number of domains per ANS. Note that few ANSes control over 600 sub-domains, while over 90% of the ANSes manage less than 4 sub-domains. I select the top N% domains to be cached by PDNS ReR and simulate attackers launching reflection attacks by pretending to experience cache misses for each domain. I assume an attacker has the list of domains supported by PDNS and a maximum upload bandwidth of 1Gbps.

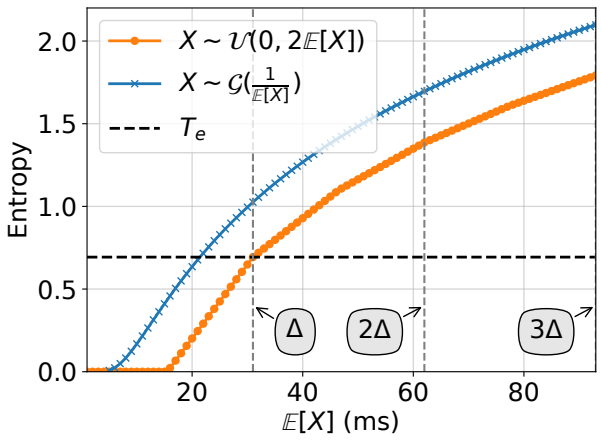
Figure 3.10(b) shows that, without the security feature introduced in § 3.4, a single attacker can generate over 100MB of traffic per second, and the traffic grows linearly as the number of attackers grows. The limit on the "reflection traffic", *i.e.,* from ANS to ReR, is dependent on the total bandwidth of all attackers and of all ANSes. When considering my security mechanism, Figure 3.10(b) shows that the reflection traffic reduces to less than 12MB per TTL, *i.e.,* for how long a record stays in the ReR's cache, with little impact of the number of attackers and the amount of domains found in the cache. This is because my security mechanism ensures that each domain can only be populated once in the PDNS ReR within its TTL, thus putting a deterministic cap on the reflection traffic. Having more attackers or fewer domains cached only allows to consume that cap faster within a TTL.

Resilience to Timing Attacks – Last, I investigate the efficacy of employing random response delays as a strategic defense mechanism against timing attacks. According to the findings in [280], 1,000 users will perform queries at an average rate of 32 QPS, which translates to $\Delta = 31$ms. The cache miss chance for each query is 0.33, *i.e.,* $m = 0.33$. Below, I calculate the entropy for a response arriving at s ms after Q_0, where s is an integer and $0 \leq s < \Delta$.

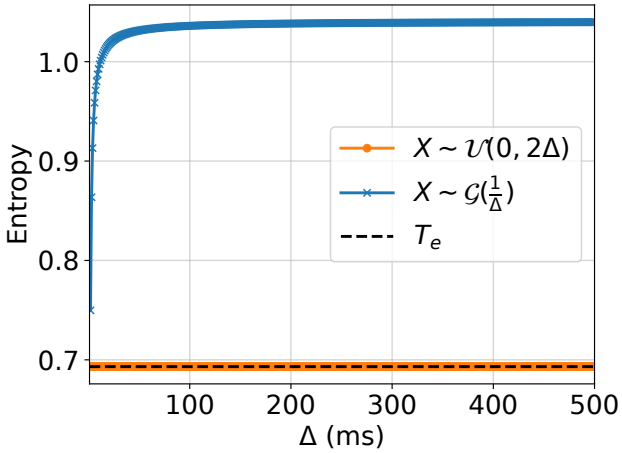
To render the outcomes more comprehensible, I introduce a threshold denoted as T_e, which



(a) Entropy as a function of s with $\mathbb{E}[X] = \Delta = 31$ms.



(b) Entropy as a function of $\mathbb{E}[X]$ with $\Delta = 31$ms.



(c) Entropy as a function of Δ with $\mathbb{E}[X] = \Delta$.

Figure 3.11: The effectiveness of delayed response forwarding to PDNS ReR.

signifies the point at which the timing attack defense is considered effective. This effectiveness manifests when PDNS ReR fails to differentiate between a response originating from two separate queries, both of which have an equal likelihood. In precise terms, I set $T_e = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2} \approx 0.69$, delineating a criterion for the defense mechanism's adequacy.

I consider two distributions in my evaluation: the uniform distribution $\mathcal{U}$ and geometric distri-

bution $\mathcal{G}$. When $X \sim \mathcal{U}(0, d)$, I have

$$\mathsf{Prob}(X_i = k) = \begin{cases} \frac{1}{d}, & k \leq d \\ 0, & \text{otherwise} \end{cases}, \tag{3.3}$$

where the average delay time is $\mathbb{E}[X] = \frac{d}{2}$. When $X \sim \mathcal{G}(p)$, I have

$$\mathsf{Prob}(X_i = k) = (1-p)^{k-1} \cdot p, \tag{3.4}$$

where the average delay time is $\mathbb{E}[X] = \frac{1}{p}$. I leave other potential distributions as future work.

First, I explore the entropy across different s ranging within the interval $[0, \Delta)$. In this experiment, I fix the $\Delta = 31$ms. For both the uniform and geometric distributions where the delay is sampled from, I adjust the parameters so that $\mathbb{E}[X] = \Delta$. Figure 3.11(a) shows that the entropy of geometric distribution is much higher than my threshold T_e, whereas that of uniform distribution is equal to T_e. It further shows that the entropy for both distributions is barely changed given different s. This means that the PDNS ReR will have roughly the same difficulty in correlating the response to the queries whenever the response arrives, illustrating the stable effectiveness of the delayed response.

Next, I investigate the relationship between the average delayed duration $\mathbb{E}[X]$ and the entropy. I fix the $\Delta = 31$ms, and calculate the average entropy across all $s \in [0, \Delta)$ for each $\mathbb{E}[X]$ value. Figure 3.11(b) illustrates that the entropy for both distributions increases as the $\mathbb{E}[X]$ increases, where the geometric distribution is overall better than the uniform distribution. When $\mathbb{E}[X] \geq \Delta$, the entropy for both distributions is greater than or equal to T_e, demonstrating the effectiveness of the defense.

Finally, I investigate how Δ affects the defense effectiveness of delayed response forwarding.

I adjust $\mathbb{E}[X] = \Delta$ and calculate the average entropy across all $s \in [0, \Delta)$ for each Δ value. Figure 3.11(c) shows that the entropy is relatively stable regardless of the value of Δ. This means that the efficacy of the defense mechanism is not notably impacted by the variation in Δ. Instead, the crux lies in the proper selection of the average delayed duration. Remarkably, the frequency at which queries are made does not exert a substantial influence on the defense's effectiveness, provided that the average delay duration is thoughtfully determined. Consequently, the query frequency primarily functions as a determinant for determining the average delay duration itself.

While my model simplifies the intricacies inherent in real-world PDNS scenarios, it undeniably underscores the potency of incorporating delayed response forwarding to PDNS ReR as a robust mechanism to counter timing attacks. In actual deployments, ANSes would opt for diverse delay-sampling distributions. Moreover, the response delay would naturally incorporate random network-related delays. This layered complexity would invariably heighten the challenge for PDNS ReR attempting timing attacks, thereby reinforcing the efficacy of the defense strategy. I leave further explorations as future work.

3.7 Related Work

DNS – In addition to the mainstream DNS systems discussed in § 3.1 and evaluated in § 3.6, privacy-aimed amendments to these proposals exist. For example, μODNS enhances the privacy guarantees of ODNS by extending the number of relays to be greater than one [217]. Di Bella et al. [156] propose a secret-sharing scheme for anonymous DNS queries which relies on a peer-to-peer proxy network. Such an approach, however, is not robust when there are too many malicious users in the system. EncDNS [184] has a similar system design to ODNS but it is less flexible in terms of the key distribution mechanism, deployability, and compatibility with the current DNS architecture. Recently, PINOT [301] proposes to obfuscate IPv4 packets in DNS traffic to random

IPv6 packets with addresses owned by the network. It manages to preserve the anonymity of users from any attackers outside of the trusted network, without modifying the DNS infrastructure. However this solution only works for users in organizational networks but not individual users. Overall, none of these solutions satisfy the privacy requirements specified in this chapter.

PIR-Based DNS Systems – Few previous works consider adopting PIR into DNS. Differently from PDNS, they require significant modifications of the modern DNS architecture. A recent efficient stateful PIR benchmarks its performance on private DNS queries [325]. However, it assumes a single PIR server storing all the DNS records, ignoring the self-governing nature of the current DNS system across all ANSes. It is also not practical from the performance perspective. Specifically, it incurs massive communication and computation overheads for all the domain owners to update their DNS records, not to mention the extra client-server communication needed whenever the cache is updated. Lu and Tsudik [229] propose a novel DNS system based on distributed hash tables and single-server PIR, where DNS records are stored in a table split into chunks which are assigned to different ReRs. A DNS client first identifies the chunk associated with a DNS record via hash indexing, and then initiates a PIR query to retrieve the record from the ReR responsible for this chunk. The main drawback of this approach is that a query reveals the "chunk" of potential DNS records it is contained in, which can be used to analyze user preferences. Previous works also consider DNS systems based on two-server PIR [284, 324], which does not offer satisfactory privacy to DNS users as discussed in § 3.1.2.

3.8 Discussion and Limitations

Limitation on Anycast – PDNS does not support *anycast* [88, 253], a network addressing and routing methodology which allows a single IP address to be shared by multiple ReRs around the world [79, 82]. This is because a ReR's cache is populated by participating final ANSes. If anycast

were to be implemented for PDNS ReR, an ANS located in a different area than the user would route the DNS response to a different ReR than the original one queried by the user. Therefore, anycast cannot be used to automatically redirect queries to the closest location, instead a unique IP address per location is needed.

Cluster of PDNS ReR – In a realistic deployment, a PDNS ReR consists of potentially multiple clusters of resolvers located at different locations. The user would select the closest location either manually or by using some self-configuring software. As this selection occurs, the user would register to a ReR which implies learning about the cache size N and compute and share pk (see § 3.2.3). I assume that servers within the same cluster (*i.e.,* one single IP) share a synchronized cache, *i.e.,* having same size and content. This implies that pk can be shared among such servers, and load balancing of both queries and DNS record updates is straightforward.

ReR In a Trusted Execution Environment (TEE) – This approach allows to verify, through a hardware vendor, that the ReR's code corresponds to an open-source non-logging implementation. Relying on TEEs [270] inherently introduces a reliance on hardware vendors, which contradicts my objective of eliminating non-collusion agreements. It also adds another point of failure, as TEEs are not free from bugs and security breaches [86]. Further, TEEs often have limited memory [13], constraining the ReR functionality. Finally, it imposes a great burden on TEE vendors while negatively impacting user experience, as every ReR attestation requires communication with an attestation server at TEE vendor.

Multi-Service Internet Companies. In some cases, companies operate diverse businesses. For instance, Google and Cloudflare simultaneously provide ReR and (final) ANS services. This positions them as critical junctions in the Internet, aggregating a wealth of information from these different operations. PDNS stands out in offering optimal joint privacy protection to users in such scenarios. Notably, only ReR-Less DoH and PDNS can effectively shield the ReR sector from

accessing both individual and regional access patterns, without the need to rely on non-collusion assumptions. The trade-off here is that ANSes receive direct queries from users, potentially revealing their identities. As discussed earlier, I deem this acceptable since this information would be accessible to final ANSes through other means. Still, I aim to minimize information exposure. ReR-Less DoH exposes 100% of the DNS queries to all ANSes. In contrast, PDNS limits this by directing only 3% of the queries to non-final ANSes and 30% to final ANSes (§ 3.6.4). Additionally, PDNS provides security guarantees to ANSes thanks to its query validation scheme (§ 3.4), which is instead not viable in ReR-Less DNS. I therefore conclude that PDNS is the preferable choice here, minimizing information exposure jointly at both ReR and ANS for multi-service companies.

Third-Party Final ANSes. Many small/medium businesses rely on third-party final ANS services such as Route 53 [2]. In this case, PDNS direct ANSes queries – in presence of cache misses – would leak some information to a third party which does not already have access to this information via direct traffic, *e.g.,* HTTP(S) in case of a webpage. The only solution to handle such privacy leak is for Route 53 to implement support for the PIR protocol used by PDNS ReR. In this setup, the PDNS ReR's cache record for a domain name would include only the IP address of the final ANS and a distinct flag, signaling users to communicate directly with the final ANS using the PIR protocol. I leave this as future work.

For small businesses, the incentive to rely on a PIR-enabled ANS is to enhance privacy assurances for their users. Such assurances could be visibly acknowledged by users through an added security icon in their browsers, providing a competitive edge over similar websites with lesser privacy protections. This model also presents an incentive for public final ANSes like Route 53, as they can put an extra charge to offset their operational expenses, thereby aligning improved user privacy with their financial objectives.

3.9 Summary

The Domain Name Service (DNS) is a fundamental component of the Internet which still suffers from privacy infringements [38] despite the recent adoption of encryption (DNS over HTTPS or DoH), and new proposals to de-associate a user identity with her query (oblivious DoH or ODoH). The main culprit of such privacy violation is the *recursive resolver* (ReR) whose role is to provide distributed caching to DNS enabling its scalability and fast speed. While some recent provocative proposals [171, 278] argue for its removal – which would naturally address such privacy concern – such approaches are unfeasible, as I have largely evaluated in this chapter. Motivated by these observations, I have proposed PDNS, to my knowledge the first practical solution to allow ReRs to operate privately thanks to the adoption of single-server Private Information Retrieval [241]. PDNS is designed to *augment* rather than replace DNS, similarly to DoH and ODoH, but with higher privacy, *i.e.,* also defending from collusion and regional access pattern analysis.

CHAPTER 4

SEMANTIC COOKIES FOR ANONYMOUS ONLINE STREAMING ANALYTICS

The ability to extract user analytics in a timely manner is of critical importance for numerous online applications [210]. An ad provider can more promptly adjust its ad layout to capture more clicks based on the user analytics extracted over short time scales. Many online services are utilizing machine learning systems to "learn on the fly" and either adjust content presentation (*e.g.*, return search results tailored towards a given user profile) or optimize system performance. Still, such machine learning systems fundamentally depend on analytics "triggers," which again, if available sooner or over short timescales, are more valuable.

Currently, the streaming analytics "machinery" typically resides in data centers. On the one hand, the analytics servers are fed by streams from web server clusters, which typically serve tens of thousands of clicks arriving per second, on average. On the other hand, given that user web requests alone are *semantic-oblivious*, *i.e.*, carrying only a personal identifier but no direct information about users, such information first needs to be obtained from associated user-profile databases. The analytics servers thus aggregate data streams from the web servers and user databases to provide advanced analytics.

This approach, however, suffers from two main drawbacks. The first drawback comes from the trend of infrastructure migration towards the network edge. In particular, content and service providers have been continuously pushing their systems and content to the users, from the content delivery networks (CDNs) to the off-nets – servers outside their own autonomous systems (ASes) – which have become a common approach to expanding the footprint of content hypergiants [175]. Nevertheless, the semantic-oblivious requests cannot be analyzed before they reach the data centers

that are distant from these edge systems/contents.

The second drawback is disrespect for user privacy, which has raised increasing attention and concerns in recent years [107, 145, 308]. More concretely, the semantic-oblivious requests, while simple in design and hence commonly adopted, carry personally identifiable information, *e.g.,* user IDs. The user IDs have allowed the service providers to record any information about the individual users as much as they can for an indefinite duration as long as the users do not actively clean up – and most users are not aware of it at all.

In this project, I explore the potential of catching and pre-processing user clicks early, much sooner than when they reach the data centers while preserving user privacy to the largest extent. In particular, I look at the content providers' network and off-nets, as well as edge ISPs. My goal is to design a system to make early click catching, in-network processing, and anonymity preserving analytics possible, and to quantify the achievable performance benefits.

To enable this approach, I propose *semantic cookies*, encrypted data structures set by the server and then kept at the user. Contrary to widely-used state-of-the-art HTTP cookies, which are effectively pointers to semantic user databases (typically hosted at data center back-ends), I plant semantic user information that is not personally identifiable directly into the cookies themselves. This enables collaborating edge components, mostly edge servers but also switches, to analytically process the user requests. Importantly, semantic cookies can be seamlessly deployed without altering any of the existing protocols.

I design and implement Snatch, the first prototype of my edge-network analytics system. I explore two designs. The first one places semantic cookies at the application layer, HTTPS, and processes them at the off-net's or CDNs' edge servers. The second one places semantic cookies at the transport layer, QUIC, which enables processing them at ISP switches. The underlying trade-off is that application-layer semantic cookies provide a high flexibility in terms of the number of

user features, while transport-layer cookies provide faster analytics. Luckily, both types of cookies could be utilized simultaneously, when needed.

Snatch is a two-tier analytics system. The first tier consists of either edge servers, which handle application-layer semantic cookies, or LarkSwitches, which handle transport-layer semantic cookies. The second tier consists of AggSwitches, which inspects all the incoming packets to the analytics server. The first-tier devices early re-direct semantic data to the state-of-the-art analytics servers. Optionally, the two tiers coordinate to enable in-network analytics. The number of supported operations available at switches is considerable; hence, it provides a valuable in-network analytics support. Snatch augments existing analytics systems in a fully cooperative manner.

I implement Snatch and evaluate it in a testbed. To fully understand the performance gains that Snatch can achieve on the Internet, I conduct a large-scale measurement study. In particular, Snatch involves several components: the edge server, ISP switch, web server, and analytics server. To study the performance of these entities in practice, I host HTTPS websites using AWS EC2 instances. In addition, I purchase CDN services from Cloudflare and AWS CloudFront. Finally, I utilize over 2,000 residential nodes from the Mysterium VPN, spread around the world, as users. These measurements enable us to accurately estimate network latencies among users, edge ISPs, off-nets and CDNs, and data centers, and evaluate performance gains achievable by Snatch.

4.1 Background And Motivation

4.1.1 Streaming Analytics

Data streaming analytics targets enormous data that arrive continuously in time. Efficient data streaming analytics is essential to many important real-time applications, *e.g.,* social networks [128], ad campaigns [140], and beyond [153]. Early streaming analytics systems use dataflow models [100, 123, 127]. With the increasing demand for streaming analytics, the last decade has

witnessed a thrive of proposals: MillWheel [91], Storm [294], Heron [215], Puma [128], Spark Streaming [65, 99, 322], Apache Flink [62], and more. Among them, Spark Streaming [322] started to aggregate the streaming data over a short interval and performs batch analytics in a Map-Reduce fashion [155]. The state-of-the-art streaming analytics produce results at a timescale of $\sim$1 to $\sim$10 seconds [73, 140, 322].

The above-mentioned work focuses on streaming analytics in a single cluster environment, leaving the *arrival of data* out of scope. In this project, I make the arrival of data a central topic of my research. For example, message queues [63, 64, 68, 75] are usually adopted in real-world production to link the data ingestion pipeline and the streaming analytics systems [140].I include the message queues when discussing the streaming analytics systems in this project.

Yet, the message queues and the streaming analytics systems do not depict the whole picture. While some applications analyze only internal data, *i.e.,* stored or generated *inside* the data center, many applications analyze data from outside the data center, *e.g.,* the users' requests, generated from end-user networks scattered around the world. Further, in online applications scenarios, the application-level streaming data is typically sent to the analytics servers only after it reaches web server endpoints in data centers. Hence, rather significant latency can be added to the user requests after they are generated by end users.

The time cost incurred before data arrives at the analytics server is nontrivial (see § 4.1.3), however, it is often disregarded. To depict a comprehensive picture, I consider an entire online streaming analytics *cycle*. The cycle includes the streaming data generation and transmission, *i.e.,* users send requests to the servers and the servers process the requests, data processing, *i.e.,* by message queues and event processors [73]. Finally, the cycle terminates with a traditionally-defined streaming analytics system, *e.g.,* Spark Streaming [65].

4.1.2 Anonymity Preserving Analytics

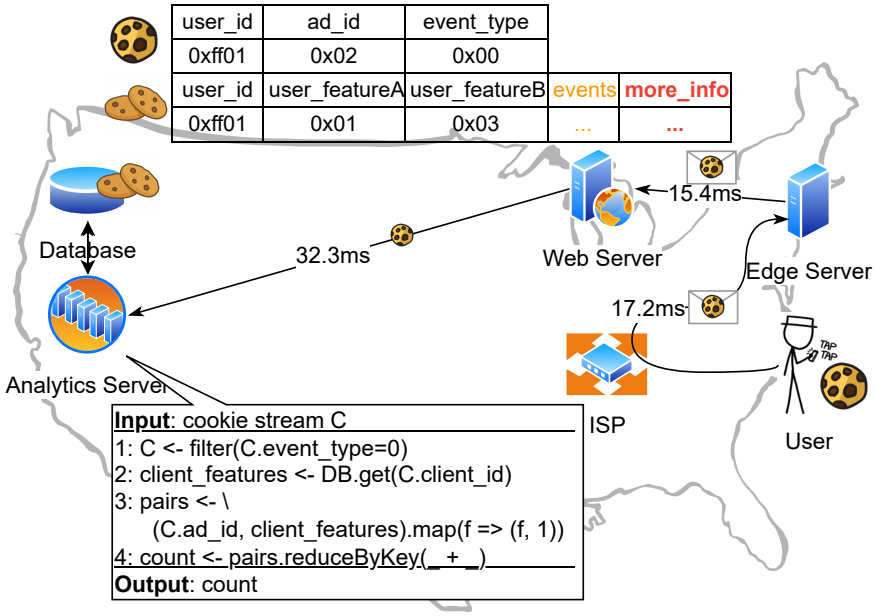
Anonymity preserving analytics refers to computational analytics over aggregated results from the users without revealing the individual identities, and hence provides strong privacy guarantees [163, 319]. Unfortunately, widely-adopted Web cookies present a significant privacy-leaking vertical, even at the network level [308]. A single identity leak in one application opens up unforeseen tracking opportunities.

With the growing public attention and concerns about individual privacy, anonymity preserving analytics has been supported by legislators [51]. The related studies have also become a hot topic in the security and privacy academic community [145, 165]. Complying with the trend, hyper-giants have also introduced their own data collection and analytics systems that preserve user anonymity, for instance, Google [107, 166], Apple [293], Microsoft [159], and more.

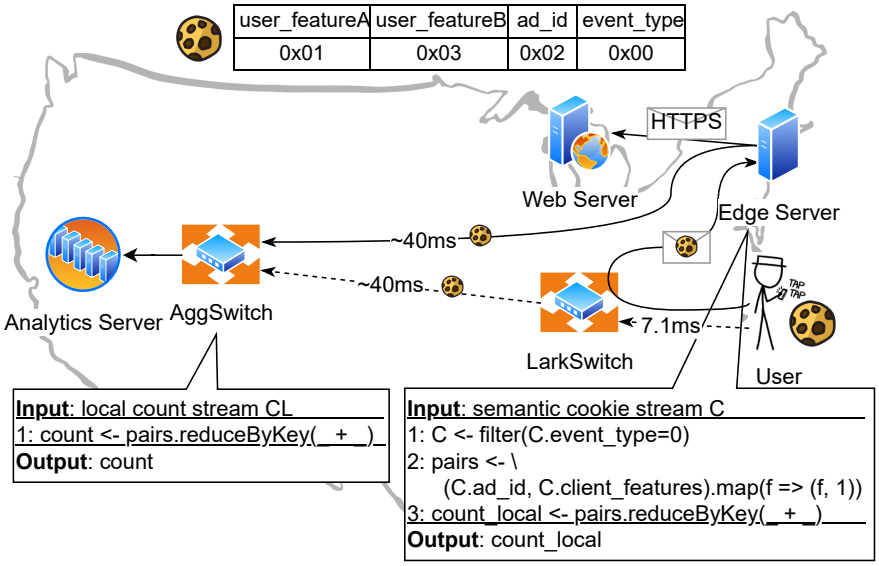
4.1.3 Opportunities

Migrating infrastructure towards the edge. Content and service providers are continuously pushing their systems closer to the users. Content delivery networks (CDNs) allow the content providers to place static content at servers nearby users, and thus improve their experience. CDNs have thus become one of the most crucial components of the Internet today, serving billions of users across the world. In fact, more than half of the Internet traffic originates from several top CDN providers [175]. In addition to building their own data centers and backbone networks [190, 214], the major content providers also deployed off-net servers [175]. Such servers are placed in the eyeball, end-user, networks. The deployment of these edge servers further reduces the latency between the user and the content, thereby improving the user experience.

In parallel with this trend, the service providers are also pushing computation closer to the users. Hence, many distributed streaming analytics systems are proposed, aiming at working with



(a) Scenario without semantic cookies.



(b) Scenario with semantic cookies.

Figure 4.1: Breakdown of time cost in a simple application of advertisement campaign.

limited resources available at the edge [121, 122, 174, 244, 260]. While helpful in certain specialized scenarios, most applications still require *centralized* streaming analytics with data from users scattered all around the globe. In this project, I focus on centralized streaming analytics.

Importantly, with the introduction of edge servers, the overall architecture of online streaming analytics systems has changed. The streaming analytics server (cluster) is usually not placed in the same region as the edge servers, which is where the users are directed first. This results in complication of the security and privacy issues, *e.g.,* a third entity has access to the cookie content, as well as a rather substantial increment of delay between the components of the online streaming analytics systems, as I will demonstrate below.

Case Study. The above infrastructure migration to the edge affects many online applications. A first example is that an advertisement provider may want to receive aggregated results of its ongoing advertisement campaign in real-time to make decisions, *e.g.,* the offering in the following advertisement auctions, based on them [140]. A second example is that real-time crowd analytics, a technique crucial to many businesses [83], needs to aggregate results about information in a particular region. A third example is the needs for faster response to users' resource demands. Today, cloud platforms have become the go-to solutions for many companies because of their capability to scale up/down in a timely manner. Nevertheless, the service scaling (where containers are usually used) needs to deploy before they become available. Hence, faster response to the demand and hence earlier provoking service deployment changes are crucial to the user experience for various online applications [50].

Below, I analyze the first example of the advertisement campaign in detail. Here, the data is generated when a user clicks on an ad link. It follows that a request is sent to an edge server, *e.g.,* in the case of a CDN, with the user ID embedded in the HTTPS cookie and the ad ID included in the HTTPS URL. The edge server then passes the cookie to the web server in the closest data

center. Next, the web server processes the cookie and delivers the data to the (centralized) analytics system which is potentially at another data center. Message queues are usually adopted to deliver the data. If the user semantic information is needed, *e.g.,* demographic or other information, the analytics server needs to first fetch this data from a database before being able to perform further analytics operations.

In this example, I assume that the application developer, who owns the web server, has control over all the cookies, meaning that they are all first-party cookies. As a result, the cookies are initially sent to the web server, and any ad broker either resides at the analytics server or receives information from it. It is important to note that in the current Web, this assumption may not hold true, as users may send separate requests to ad broker URLs along with third-party cookies. However, the use of these third-party cookies contradicts the prevailing trend of enhancing privacy and has already been banned by some major browsers [55, 57, 85] and is expected to be banned by the remaining browsers in the near future [53]. Consequently, I focus on first-party cookies in this study.

Drawbacks and Opportunities. I conduct a large-scale measurement study to comprehensively quantify the latency inflation cost (details are provided in § 4.4). Figure 4.1(a) illustrates an example of the analytics time cost breakdown of one data point. It starts from the request generated by the user in New York[1]. The closest edge server, which caches static content, is selected and is in New York. The web server that provides dynamic content and handles cookies is hosted at AWS's *us-east-1* region. The server for global streaming analytics is, however, located in California. All QUIC connection handshakes take 97.8 ms in total. Adding up to the processing time costs at both the edge and web servers, which take 378.2 ms in total, as well as the delay of 32.3 ms from the

[1]Here I assume QUIC is adopted as it is becoming popular, and its handshake is simpler than TCP+TLS. With TCP+TLS handshakes, the time cost of communication in Figure 4.1(a) will be more than doubled as it is now.

web server to the analytics server[2], the time cost before the cookie arrives at the analytics server is 508.3 ms, which occupies more than 50% of the total time cost assuming 500 ms is needed for the analytics itself[3].

Besides the latency inflation highlighted above, I also notice that in the current online streaming analytics systems, the analytics operations are performed only after they arrive at the analytics server. For instance, traversing through the path of the data in Figure 4.1(a), I find that the data is held by the edge server and web server for more than 300 ms in total while they respectively handle the static and dynamic web request – unrelated operations to the data analytics – and is left untouched before it arrives at the analytics server.

I now move to privacy issues. In short, the above request submits a user activity, *i.e.,* clicking an ad, along with the user ID to both the edge server controlled by a CDN provider, and the analytics server controlled by the ad provider. After that, the ad provider can perform any analytics as it wishes, or save this event associated with the user ID in its database for further analysis. Yet, this might lead to potentially serious privacy violations. This is because as long as the user does not clean up the cookie, all her activities will be logged among potentially other individual information that is obtained through other sources, as illustrated by the tables in Figure 4.1(a). An attacker, *e.g.,* a malicious data owner or a third-party attacker who gets access to the database of the ad provider, or network traffic, might then be able to impose danger to such individual users by splicing all the information pieces [308].

[2]I assume no handshake is needed because a persistent connection is established by the message queues, or otherwise the time cost will be greater.

[3]The default computing time interval of Spark is 1s [78]. If the data arrives evenly, the average time of analytics is 500 ms.

4.2 System Design

In this section, I first present the overview of Snatch design and illustrate the benefits using the same example as in § 4.1.3. Next, I present my security and privacy threat model. I then present more design details and benefit quantification of the semantic cookie as well as in-network streaming analytics. Later, I touch on the functionalities of Snatch's controller. Finally, I detail the security and privacy guarantees of Snatch.

4.2.1 Overview

At a high level, I propose to forward and (pre-)process the data much earlier than the current online streaming analytics systems do – at the edge server, or even at the ISP switch, thanks to the programmable data plane [110]. One critical obstacle for the early data forwarding and pre-processing is that the cookies are semantic-oblivious, *i.e.,* no information about the user but only a reference to the information is included. This is because at the time when the cookies are assigned, the server knows nothing about users. I instead propose *semantic cookies*. Like regular application-level cookies, they are generated by servers, and kept by users. The difference is that semantic cookies hold encrypted application-level user data, and more importantly, include no personally identifiable information. Typically, once the information about the user is collected, *e.g.,* when a user clicks a specific web page, the web server should push semantic information into the user cookie itself. To the best of my knowledge, I am the first to seize this opportunity, given the nature of Snatch.

The procedure of Snatch is illustrated in Figure 4.1(b), focusing on the same application as the case study in § 4.1. A previous benchmark study [140] evaluated the streaming analytics engines by operating a join operation to obtain the number of users who viewed the ads. But in practice,

more advanced analysis of the composition of the users may be needed to allow the ad providers to make decisions based on the results. Thus, I assume that the analytics server wants to analyze the composition (by their demographic categories) of users who viewed a particular ad in an instant windowed time. This can be achieved with three operations as shown in Figure 4.1(a): (i) filtering the arriving cookie streams by event type, *e.g.,* a user viewed an ad (L1); (ii) then requesting the database for the user features (demographic information) by the user ID embedded in the cookies (L2); and (iii) counting the number of users for every user feature (L3-4).

In Snatch, the web servers should set the semantic cookies as a replacement of the user ID, as shown in Figure 4.1(b), after the first connection with the user and the information of the user becomes available. It is noteworthy that the first connection cannot be accelerated and is not depicted in Figure 4.1; all the results I present in this project focus on subsequent connections after the initial one. The semantic cookies should be kept by the user, similar to the current design. What is different is that the ad provider should not store any user information. From then on, the user sends requests with the semantic cookies. The semantic cookies can be recognized and processed by the edge server. As shown in Figure 4.1(b), the edge server filters the cookies by the event type (right L1). It also counts locally the number of users who viewed a particular ad for every user feature (right L2-3). The processed data can be forwarded directly to the analytics server. Before it arrives at the analytics server, a programmable switch close to the analytics server named AggSwitch aggregates the local counts from all the edge servers (left L1) before delivering it to the analytics server.

As there are many devices and parties involved in Snatch, a controller (not shown in Figure 4.1) is present to coordinate all the participants. As shown in Figure 4.3, Snatch controller is run by a trusted party. It accepts analytics tasks from application developers and distributes the associated instructions to different devices held by different parties.

In this example, all the analytics have been completed on the way to the analytics server while no user ID is present. The time costs on analytics operations ($\sim$500 ms) are reduced to <1 ms given that (i) each web server only handles a small number of requests and hence has minimal costs and (ii) the line-rate processing ability of the programmable switches. It follows that the total latency from when the data is generated to when the decision can be made based on the data is reduced by $\sim$80% from 1008.3 ms to 228.6 ms. This demonstrates the feasibility and benefits of processing the data early.

Moreover, the semantic cookies in many scenarios are constant. For instance, in the second example of real-time crowd analytics, what needs to be aggregated and analyzed is the user's information, *e.g.,* demographic or interests; in the third example of faster response to users' resource demands, what needs to be aggregated and analyzed is the typical demand of the users. These information can be kept at the user's side and sent without knowing what the user's requests are. Thus, I further propose to encode encrypted semantic cookies in the transport layer. With the programmable switch's capability to read and parse packet headers, the semantic transport-layer cookies can be acted upon as soon as the user requests reach the edge ISPs, as the dashed lines in Figure 4.1(b) illustrate. In particular, semantic cookies could (optionally) be pre-processed, and forwarded by the LarkSwitch, as shown in the figure. This further cuts analytics latency to around 48 ms – a $\sim$95% reduction in the total delay.

4.2.2 Threat Model

I assume a third-party attacker who can monitor and collect network packets from a limited geolocation range. I also assume an attacker who may join the system as a user to receive the semantic cookies from the web servers. The attacker may try to decode the format of either the application-layer or transport-layer semantic cookies by examining the collected packets. Nevertheless, the

attacker should be computationally bounded and not be capable of decrypting ciphertexts that are encrypted using advanced cryptography algorithms, such as AES and TLS.

In general, being able to decode the semantic cookie would allow the third-party attacker to intercept user information from network eavesdropping, or send fake data [137, 176, 231, 234] to distort the application developers' analytics results.

Moreover, I assume an honest-but-curious edge node, *i.e.,* edge server or LarkSwitch in Figure 4.1(b), who follows the protocol but may try to understand the application-layer purposes of the semantic cookies, and hence steal the user information for commercial purposes. On the other hand, I assume a malicious application developer who may try to insert personally identifiable information into semantic cookies while using Snatch.

4.2.3 Semantic Cookie

Contrary to "traditional" cookies, which are used as pointers to a back-end database of user attributes, semantic cookies enable web servers to directly encode user attributes and push them to the end-users. The semantic cookies cannot be in plaintext but need to be encoded and encrypted because they will be stored on the users' side.

Application-Layer Semantic Cookie. Because the edge server is the endpoint of the users' TLS connections, it has the access to all the application-level information in the users' requests, including the application-level cookies as required by Snatch. For instance, if users are sending an HTTPS request, then the edge server is able to access the headers, cookies, and payload of the HTTPS request.

Therefore, leveraging edge servers for early forwarding the application-layer cookies is straightforward to implement. Most current edge services, *e.g.*, Cloudflare's CDN, allow the user to set custom page rules to adjust caching levels, forward requests, modify headers, *etc* [66, 67]. What

is needed in Snatch is to decrypt the cookies, match semantic cookies' names and values, and send the extracted data to a custom destination (analytics server) – if possible – in a custom format. The additional computational cost is minimal as it is similar to existing header-related operations.

The benefits of application-layer cookies are three-fold: First, they can support semantic cookies with as many sub-cookies (user features) as needed by the applications. Second, they do not require any modification on the user's side. Third, they are fully compatible with the current HTTPS request design and simply need to include semantic cookies. In addition, the cookies can be easily kept across different connections between the user and the server over time, regardless of the underlying protocol, *e.g.*, TCP, UDP, QUIC, TLS, *etc*.

To better quantify the benefits of using semantic versus non-semantic application-layer cookies, I aim to quantify the speedup. It is defined as the ratio of the expected latency in two scenarios, *i.e.*, non-semantic vs semantic. Hence, speedup is $\geq$1. Denote user by C, edge server by E, web server by W, and analytics server by A. Then, d_{CE} is the delay between the user and the edge server, and so on. Let T_{trans} be the transmission duration of the request.

I further denote by T_E, T_W, and T_A the time costs for processing requests at the edge server, web server (including database communication), and analytics server (including message queues), respectively. Then, for HTTPS request on top of QUIC 1-RTT, the speedup is

$$S_{app-https} = \frac{3d_{CE} + 3d_{EW} + d_{WA} + T_{trans} + T_E + T_W + T_A}{3d_{CE} + d_{EA} + T'_E + T'_A}, \tag{4.1}$$

where T'_E and T'_A are the time costs when Snatch is involved. Because of the minimal additional cost from processing application-layer cookies at the edge server, I consider $T'_E = T_E$. Further, coefficient 3 in the equation comes from the QUIC 1-RTT handshake process.

As far as QUIC 0-RTT is concerned, because it sends data at the very beginning, I have

$$S_{app-https-0rtt} = \frac{d_{CE} + d_{EW} + d_{WA} + T_{trans} + T_E + T_W + T_A}{d_{CE} + d_{EA} + T'_E + T'_A}. \tag{4.2}$$

I further look into the speedup for application-layer semantic cookies when TCP connections are adopted. For an unencrypted HTTP request, on top of TCP, the speedup S_{app} of the streaming analytics is

$$S_{app-http-tcp} = \frac{3d_{CE} + 3d_{EW} + d_{WA} + T_{trans} + T_E + T_W + T_A}{3d_{CE} + d_{EA} + T'_E + T'_A}, \tag{4.3}$$

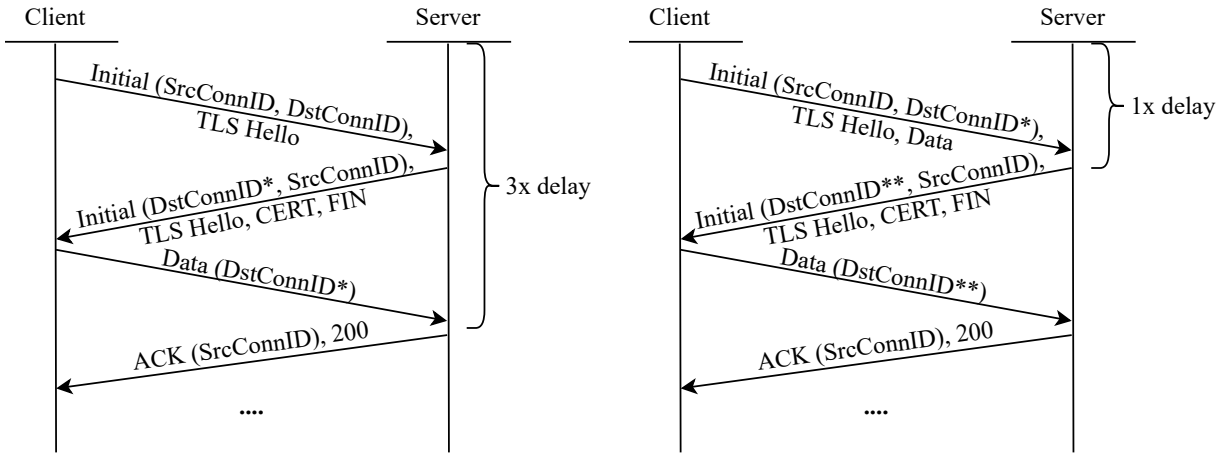
where the coefficient 3 in $3d_{CE}$ and $3d_{EW}$ comes from the 1-RTT TCP handshake process during the connection establishment.

For HTTPS requests, TCP + TLS 1.2 handshakes need at least 3 RTTs to set up. Thus, the speedup is

$$S_{app-https-tcp} = \frac{7d_{CE} + 7d_{EW} + d_{WA} + T_{trans} + T_E + T_W + T_A}{7d_{CE} + d_{EA} + T'_E + T'_A}. \tag{4.4}$$

For example, 3 RTTs needed to establish an HTTPS connection between a client and an edge server implies 7 one-way delays, *i.e.*, 7 d_{CE}.

Transport-Layer Semantic Cookie. Transport-layer cookies are semantic cookies that are encoded in the transport-layer protocol. As identified in a previous study [102], there are three ways to encode cookies in the transport layer without requiring any modifications on the users' machines: (1) encode the cookie into the least significant bits of IPv6 addresses with a maximum of 64 bits, (2) encode the cookie into the `timestamp` option of TCP with a maximum of 32 bits, and (3) encode the cookie into the `conneciton ID` of QUIC with a maximum of 160 bits.



(a) QUIC 1-RTT. (b) QUIC 0-RTT.

Figure 4.2: QUIC handshake procedure and the time cost for the server to receive data.

IPv6 – The use of IPv6 addresses requires the assumption that the MAC address is associated with the least significant bits of the IPv6 address, and thus is not appropriate in my case. I consider the other two options: via the TCP timestamp and via the QUIC `conneciton ID`.

TCP – When the TCP timestamp option TSP is set and used in one direction (*e.g.*, from server to client), all the packets in the reverse direction (from client to server) will attach the same TSP value automatically. However, there are several issues with this approach. First, the TSP value cannot be reused in the next TCP connection. Second, if the client wants to send the cookie in the next TCP connection proactively, it requires non-negligible modification on the client's side – access to the root privilege and modifying the outgoing packets accordingly. This breaks my vision of minimal to no client modification.

QUIC – QUIC is a transport-layer protocol implemented in the userspace on top of UDP. The QUIC connection establishment procedure is illustrated in Figure 4.2 (left for the 1-RTT handshake, right for the 0-RTT handshake). For QUIC 1-RTT, a long QUIC header will be used during the handshake phase. The client will send two randomly generated connection IDs *SrcConnID* and

DstConnID. Then the server will copy *SrcConnID* but set a new *DstConnID** and return them to the client. In the following communication, a short QUIC header will be used where the client sends packets with *DstConnID** and the server sends packets with *SrcConnID*. Further, the server can reset the connection ID with version negotiation packets at any time. For QUIC 0-RTT, it is only applicable when there was a previous connection between the same end-points. The client will send the same *DstConnID** as in the last connection. An IETF proposal [162] has also suggested the potential of QUIC `conneciton ID` other than identification for the connection.

I find that the `conneciton ID` field, in particular *DstConnID**, allows the encoding of transport-layer cookies. In addition, it takes minimal effort to modify the `conneciton ID` field because QUIC is implemented in the userspace. Thus, it fits my vision of minimal (QUIC-1RTT) to no (QUIC-0RTT) client modification.

Snatch fully utilizes the features of QUIC. I consider all the connections between a user and an edge server except the first one – at least one connection is needed before semantic cookies are available. If the user uses QUIC 0-RTT, she repeats the connection ID from the last connection where transport-layer cookies are encoded. LarkSwitch then will be able to decode the transport-layer cookies and forward them to the analytics server. This requires no modification on the user's side. If the user uses QUIC 1-RTT, a slight modification of the code in userspace is needed to allow the QUIC 1-RTT to keep the transport-layer cookie in the new connection, *i.e.,* QUIC should remember the connection ID from last connection but re-generate a subset of the bits without tweaking the transport-layer cookies. In summary, both QUIC 0-RTT and 1-RTT fit my vision and work for Snatch.

I further quantify the benefits of transport-layer semantic cookies. Let I denote ISP. Hence d_{CI} is the delay from user to ISP. Similar to the analysis for application-layer cookies, the speedup of

the streaming analytics for QUIC 0-RTT is

$$S_{trans-0rtt} = \frac{d_{CE} + d_{EW} + d_{WA} + T_{trans} + T_E + T_W + T_A}{d_{CI} + d_{IA} + T'_A}. \tag{4.5}$$

For QUIC 1-RTT, its handshake needs 1 RTT and hence the coefficients for d_{CE} and d_{EW} become 3, while the denominator keeps the same as the transport-layer cookie is included in the first packet header. The speedup is

$$S_{trans-1rtt} = \frac{3d_{CE} + 3d_{EW} + d_{WA} + T_{trans} + T_E + T_W + T_A}{d_{CI} + d_{IA} + T'_A}. \tag{4.6}$$

4.2.4 In-Network Streaming Analytics

Snatch further seizes the opportunity to accelerate streaming analytics by leveraging the in-network computation: the programmable switch performs computation at line rate, much faster than the servers [259]. For transport-layer cookies, streaming analytics can be completed in the data plane – via the cooperation of LarkSwitches and the AggSwitch. The LarkSwitch decodes the transport-layer cookies, pre-processes the data, and send them to the analytics server. On the last hop to the analytics server, an AggSwitch extracts and aggregates the data from all LarkSwitches. Note that for application-layer cookies, the analytics can be done in the network as well. It only requires the edge server to forward the application-level data in a format agreed in advance, which allows AggSwitch to decode and aggregate the data.

The modern programmable switch is able to perform AES encryption/decryption [136] and calculate most of the common statistics [195, 226]. I limit the pre-processing to the supported operations, and leave more complex ones to the analytics servers. When all the operations of a target analysis are supported by the switches, Snatch reduces all the time costs of Pub/Sub services

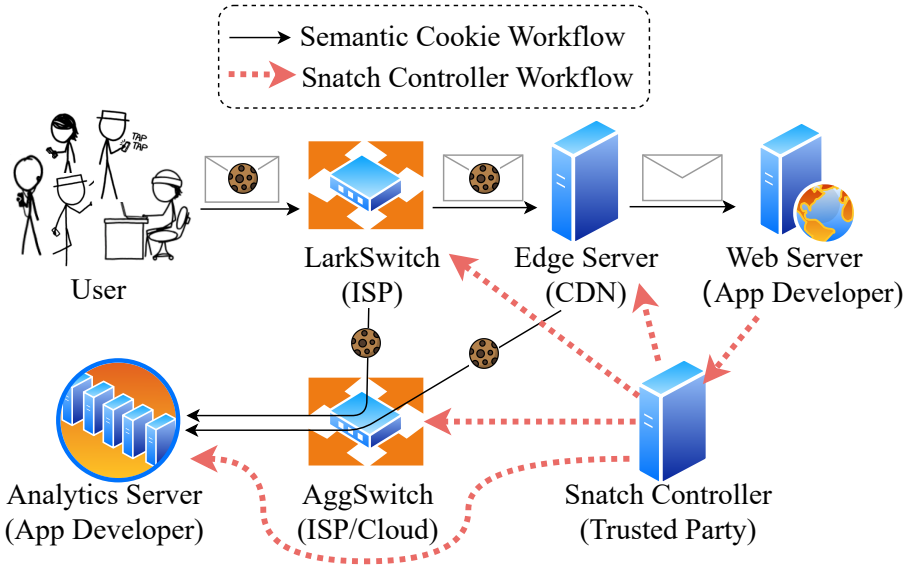


Figure 4.3: Snatch controller workflow.

and the analytics process.

I consider two types of forwarding schemes: per-packet and periodical forwarding. Per-packet forwarding satisfies the needs of applications that require very low latency and immediate knowledge of the streaming data. When all the operations of a target analysis are supported, Snatch provides a huge speedup, *i.e.,* $T'_A < 1$ ms because the programmable switch operates at line rate. On the other hand, periodical forwarding targets applications that have slightly loose requirements on latency. During each period, the programmable switches updates the statistics based on incoming packets. By the end of each period, LarkSwitch and AggSwitch cooperate to calculate statistics and forward them to the analytics server. Compared to per-packet forwarding, periodical forwarding saves bandwidth resources while sacrificing latency. I explore this trade-off experimentally in § 4.4.2.

4.2.5 Controller

Snatch includes many components that spread across the current Internet infrastructure. It is not practical for any single party to possess or control all Snatch components. Instead, Snatch should leverage the existing Internet infrastructure to the largest extent and builds on top of it. To realize that, Snatch needs a controller to coordinate all the other system components. Snatch controller should be run by a trusted party that builds up commercial relationships with the application developers, edge ISPs, content providers, and optionally cloud providers. As illustrated in Figure 4.3, the application developer submits the analytics tasks to the Snatch controller, which then parses the tasks and distributes the instructions to different devices controlled by different parties, including the LarkSwitch by ISPs, edge servers by content providers, AggSwitch by ISP or cloud providers, and analytics server by the application developer or ad broker. It also returns the format of processed cookies to the analytics server which is controlled by the application developer.

At a high level, Snatch controller provides the following APIs to the application developers: (1) Add or remove applications. In my design, the system supports multiple applications at the same time. Different applications are distinguished by *application ID* in the cookies (either at transport or application layer). (2) Add or remove cookies. For each application, Snatch supports multiple user features, or sub-cookies. A transport-layer cookie is preferred if there is enough space. When the space is scarce for all the sub-cookies, the developer should decide which sub-cookies are encoded at the application layer based on the needs. (3) Change feature type and valid ranges. Snatch supports two types of data: *class* and *number*. For a different feature type, Snatch supports different pre-processing functions. Any data that is not in the valid feature range will be aborted. (4) Change the forwarding scheme, either per-packet or periodically.

4.2.6 Security and Privacy

Snatch provides security and privacy guarantees. Following my threat model, this guarantee needs to hold against three potential attackers as detailed below.

Malicious Third-Party Attackers. First, I need to protect the cookies from being understood or tempered by third-party attackers who monitor and collect network packets. To achieve this, I propose to encrypt the transport-layer semantic cookies with AES-128 (see § 4.3), and use HTTPS when accessing the Web protecting the application-layer semantic cookies with secure communication. In this way, third-party attackers cannot decrypt or learn the format or the content of the semantic cookies. The AES encryption keys should be set differently in different regions and changed regularly to strengthen security protection. It is noteworthy that these protections are not in conflict with existing methods for mitigating data pollution [230, 232, 233, 261], which can be still applied to safeguard the integrity and accuracy of analytical outcomes.

Honest-But-Curious Edge. Next, I need to prevent the edge nodes from being able to understand the application-layer purposes of the semantic cookies – unfortunately, they can do so now. To achieve that, the app developer should avoid using semantic names and, if possible, add transformations to the values, *e.g.,* performing reversible mathematical operations before pushing semantic cookies to the users and recovering them after receiving aggregated results from Snatch. This process renders plaintext semantic cookies semantically incomprehensible. Further, app developers can set multiple correlated cookies to substantially raise the bar for interpreting them. For example, they can set two cookies to represent the same purpose, but each time only update either one of them, hence confusing the edge nodes.

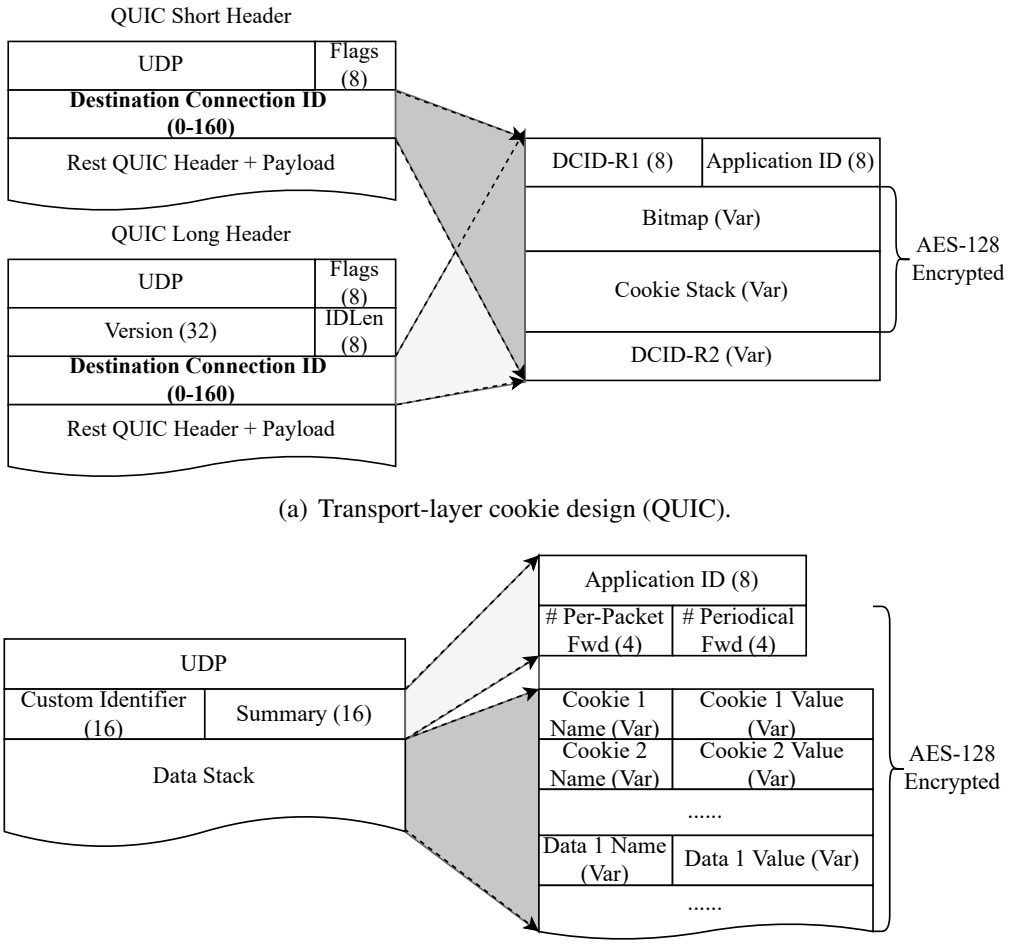
More importantly, full protection can be achieved using differential privacy (DP) [164], which secures data privacy by introducing noise to it. A naive example is that if the app developer intends

to increase a cookie value by 1, they could instead increase it by 2 with a probability of 75% and decrease it by 2 with a probability of 25%. Such noises can also be applied to all fields – including those that are not undergoing actual changes – as well as their initial values, to better disguise the sensitive data changes from specific user groups or operations. Consequently, while individual data may not be entirely precise, the aggregated analytical results such as calculated statistics from all users remain accurate.

Note that the actual DP model will be adaptive and more complex than the above example. Striking the right balance between accuracy and the level of added noise is crucial when implementing DP. But deriving specific DP models is beyond the scope of this study as they depend on the specific user data range and analytics accuracy requirements following business models.

In general, Snatch may require the app developer to redesign cookies mostly because of the discarding of personally identifiable information. Further, the app developer should leverage cookie encodings, correlation, and potentially DP, and maybe employ multiple edge providers to prevent them from learning the semantic cookies. Beyond cookie redesigns, it does not rely on any particular constraints on app developers or ad brokers.

Malicious Application Developer. The last concern is that it is possible for the application developer to include personally identifiable information in a non-semantic cookie, *i.e.,* not processed by Snatch, during the communication between the web server and the users. This is prohibited by Snatch's policy and penalties will be applied once discovered. I leave the technical enforcement of excluding personally identifiable information in this scenario as future work. It is noteworthy that while such technical enforcement is not included in this project, Snatch has made it possible to regulate the usage of individual identifiers in the cookies by providing an alternative system that works well – or even better – without such identifiers.



(a) Transport-layer cookie design (QUIC).

(b) Custom aggregation packet design.

Figure 4.4: Transport-layer cookie and custom aggregation packet design.

4.3 Implementation

I have implemented a prototype of Snatch, and I will share the details of my implementation in this section.

4.3.1 Cookies and Programmable Switch

I implemented a prototype of LarkSwitch and AggSwitch based on an Intel Tofino switch. I first present the cookie and packet design. Then, I introduce the switch logic. Further, I introduce my implementation and discuss the scope of analytics with programmable switches.

Transport Cookie Design. I choose QUIC protocol as the carrier of transport-layer cookies because it fully meets the requirement of Snatch (see § 4.2.3). I encode the transport-layer cookies in the up-to-160-bit `connection-ID` field of QUIC headers. As shown in Figure 4.4(a), I split the connection-ID into four parts: (1) 8-bit destination connection ID (`DCID`), (2) 8-bit `application-ID`, (3) `bitmap` of variable length, and (4) `cookie-stack` of variable length. DCID is randomly generated for connection identification. The application-ID is used for distinguishing from normal QUIC packets and specifying the format of the remaining bits. Because of the limited space, the format of bitmap and cookie-stack are not fixed but application-dependent. Assuming there are N sub-cookies used by an application, corresponding to N features, then the bitmap has N bits where each bit denotes whether this sub-cookie is present. The cookie-stack includes N sub-cookies and the length of each sub-cookie is pre-defined by the controller. N is bounded by the memory and stage limitation of the switch. The remaining bits (if any), unoccupied by the bitmap and cookie-stack, marked as DCID-R2 in the figure, are also randomly generated for connection identification.

I create a custom packet header on top of UDP to carry early-forwarded cookies or pre-processed data (either by LarkSwitch or edge server) for AggSwitch. Figure 4.4(b) shows that the custom packet header includes three parts: 1) a 16-bit special string `SID`, a custom identifier for distinguishing from regular UDP packets; 2) a 16-bit `summary` that contains application-ID and the number of sub-cookies/data for either per-packet forwarding or periodical forwarding, re-

spectively; 3) `data-stack` that contains N sub-cookies and data. All data after the application ID are encrypted using the AES-128 algorithm.

The extracted cookies and data encoded in the custom aggregation packet from LarkSwitch and edge server to AggSwitch may be lost because UDP is used. I argue that the benefits of using UDP overtake the loss. The loss here is that less than 0.01%, *i.e.,* the packet drop rate in today's WAN [70, 71, 77], of the cookies or data will be lost. In comparison, there are two major benefits. First, for short-term analysis, which is the target for Snatch, the value of the data is much higher when the data is available sooner. Dropping one data point out of tens or hundreds of thousands will not make a large difference to the distribution of the data, and thus to the results. At the same time, the data is not lost forever. For a long-term analysis, full and accurate results can be obtained by syncing up the records at the web servers or related databases. Second, implementing a retransmission mechanism on programmable switches is non-trivial and consumes scarce DRAM resources to keep the status. Instead, the resources can be used to offload more computation and thus provide better speedup or support more applications. In conclusion, it is the best choice for Snatch to adopt UDP for the custom aggregation packet.

To prevent the cookies and data from being hacked or tempered by the users or attackers, the transport-layer cookies after application-ID is encrypted using AES-128. The AES-128 key is only known to the application developer and the edge nodes, *i.e.,* edge server or LarkSwitch/AggSwitch. It is noteworthy that encrypting or decrypting the up-to-160-bit transport-layer semantic cookies using AES-128 only adds $\sim$0.1 ms delay with a modern Tofino switch [136].

Switch Logic. When a new application is registered at a LarkSwitch or AggSwitch, its parameters – including the application-ID, the format of bitmap and cookie-stack, and the AES key – are stored in the switches' match-action table entries. LarkSwitch will try to match the application-ID for all the incoming QUIC packets. When a packet is matched, the switch decrypts and decodes

the available cookies/data following the parameters of the corresponding application. the switch then performs counting or other statistical operations on the decoded cookies/data. For per-packet forwarding cookies or for periodical forwarding cookies when the period ends, the switch creates a new custom packet and sends it and associated statistics to the analytics server. To do so, I make the switch `clone` the original packet. The original packet is still forwarded to the web server to keep the original communication. Meanwhile, the cloned packet header will be rewritten and its payload will be removed before being sent to the analytics server.

Statistics Calculation. Both the LarkSwitch and the AggSwitch involve statistics calculation when they process the cookies and data. The match-action pipeline design makes programmable switches naturally classifiers and counters. In my prototype, I have implemented the basic statistics which satisfy the (partial) needs of most streaming applications. For data type *class*, I implement counting by matching value. For data type *number*, I implement sum, min, max, and average calculations.

I further discuss the scope of applications supported by Snatch's in-network streaming analytics. P4 switches support most of the streaming analytics operations. Here, I take Spark Streaming as a comparison to illustrate what can be done for the in-network streaming analytics (INSA). Indeed, INSA is not as flexible as Spark Streaming because of the constraint on the programming model and computational and storage resources. My goal for INSA is to assist with the streaming analytics and potentially complete relatively simple tasks alone, but not to entirely replace Spark Streaming.

While Snatch handles multiple tasks, here I focus on the "depth" of each task and hence assume to support only one task. In the discussion of the feasibility to achieve a function, I consider that modifications can be made at either the compiling phase, *i.e.,* modifying the P4 code, or Snatch application submission phase, *i.e.,* the application developer encodes the cookies and sets up the

Table 4.1: Supported operations and related application with in-network streaming analytics. N/A for not applicable, N for not supported, Y for supported, and Y* for supported with limitation.

DStream Method	**INSA**	**Category**
cache()	N/A	DStream-specific
checkpoint(interval)	N/A	DStream-specific
cogroup(other[, numPartitions])	Y*	partition, table-join
combineByKey(createCombiner, mergeValue, ...)	Y*	foreach
context()	N/A	DStream-specific
count()	Y	reduce
countByValue()	Y	reduce
countByValueAndWindow(windowDuration, ...[, ...])	Y	window, reduce
countByWindow(windowDuration, slideDuration)	Y	window, reduce
filter(func)	Y*	foreach
flatMap(func[, preservesPartitioning])	Y*	partition, foreach
flatMapValues(func)	Y*	foreach,
foreachRDD(func)	Y*	foreach
fullOuterJoin(other[, numPartitions])	Y*	partition, table-join
glom()	N/A	DStream-specific
groupByKey([numPartitions])	Y	partition, reduce
groupByKeyAndWindow(windowDuration, ...[, ...])	Y	partition, window, reduce
join(other[, numPartitions])	Y*	partition, table-join
leftOuterJoin(other[, numPartitions])	Y*	partition, table-join
map(func[, preservesPartitioning])	Y*	partition, foreach
mapPartitions(func[, preservesPartitioning])	Y*	partition, foreach
mapPartitionsWithIndex(func[, ...])	Y*	partition, foreach
mapValues(func)	Y*	foreach
partitionBy(numPartitions[, partitionFunc])	N	partition
persist(storageLevel)	N/A	DStream-specific
pprint([num])	N/A	DStream-specific
reduce(func)	Y*	reduce
reduceByKey(func[, numPartitions])	Y*	partition, reduce
reduceByKeyAndWindow(func, invFunc, ...[, ...])	Y*	partition, window, reduce
reduceByWindow(reduceFunc, invReduceFunc, ...)	Y*	window, reduce
repartition(numPartitions)	N	partition
rightOuterJoin(other[, numPartitions])	Y*	partition, table-join
saveAsTextFiles(prefix[, suffix])	N/A	DStream-specific
slice(begin, end)	Y	window
transform(func)	Y*	foreach
transformWith(func, other[, keepSerializer])	Y*	foreach
union(other)	Y*	table-join
updateStateByKey(updateFunc[, ...])	Y*	foreach
window(windowDuration[, slideDuration])	Y	window

corresponding receiver at the analytics server.

In addition, because today's P4 model only supports partial integer operation (see "Statistics Calculation" in § 4.3.1), I limit the following discussion in the scope of integer operations. Yet, it is noteworthy that it is possible to perform counting operations for strings: The application developer can either encode the string to integer or use a dictionary when the value possibility is limited. In this way, counting can be done by matching the hash value or the keyword. Still, other string functions such as concatenate are not supported. It is also noteworthy that the latest study has demonstrated that it is viable to perform float operation with programmable switches by carefully rescheduling the computation procedure [321]. An alternative is to leverage float number quantization [177].

A Spark Streaming program often executes a series of DStream methods [74], *e.g.,* map, reduce, etc, to a DStream object, *i.e.,* the data within an interval. For the sake of convenience of discussion, I classify the DStream methods into several categories: DStream-specific, partition, foreach, window, table-join, and reduce. A method may belong to multiple categories at the same time. For instance, `reduceByKeyAndWindow` belongs to three categories: partition, window, and reduce. Table 4.1 lists all the DStream methods, whether they can be done with INSA, and their categories. Indeed, the complexity of some DStream methods heavily depend on the input functions, and whether INSA supports such a DStream method depends on the input function, *i.e.,* when the operands in the input function are supported by programmable switches, the DStream method is supported by P4, and vice versa. Moreover, the total number of DStream methods that are operated on a DStream object is restricted by the limited number of pipeline stages of the programmable switches [87]. Below, I discuss the methods in detail by category.

DStream-specific methods include `cache`, `checkpoint`, `context`, `glom`, `persist`, `pprint`, and `saveAsText-Files`. They are not applicable to INSA because they are specific for assist-

ing the Spark programming model but not computation-related operations.

Direct partition methods include `partitionBy` and `re-partition`, whereas indirect partition methods, *i.e.,* where partition number is an optional input parameter, include methods in foreach, window, table join, and reduce categories. To investigate these methods, I first need to understand more about the underlying data model of Spark Streaming. Resilient Distributed Dataset (RDD) includes all the streaming data within a batch interval from all partitions, which refers to the data stored at one Spark node and is the basic operable unit in Spark. In Snatch, each edge node, *i.e.,* ISP switch or edge server, can be regarded as a partition where data is stored. But unlike Spark, the data in each partition depends on client location and activities, and cannot be moved or reassigned in Snatch. Therefore, partitionBy and repartition are not supported by INSA. However, operations on the partition are possible: AggSwitch can set up a match table for each edge node and perform different actions accordingly. The modifications should be made at the compiling phase.

Foreach methods include `combineByKey`, `filter`, `foreachRDD`, `map`, `flatMap`, `flatMapValues`, `mapPartitions`, `mapValues`, `mapPartitionsWithIndex`, `transform`, `transform-With`, and `updateStateByKey`. The main purpose of these methods is to allow operations at a finer granularity, *i.e.,* at per data point level. In INSA, the programmable switch is processing at per-packet granularity. Therefore, foreach methods are naturally supported by INSA while subjected to input function, *i.e.,* as long as the input function is supported by INSA, the foreach methods are supported by INSA.

Direct window methods include `slice` and `window` whereas indirect window methods, *i.e.,* where window settings are optional input parameters, include methods in reduce categories. Method window provides flexibility by allowing the user to extract a new windowed DStream based on the existing DStream but with a different interval. Method slice is similar but only needs aggregated

data within one interval. The periodical forwarding in Snatch is similar to window methods as it returns data on windowed packets. In the same spirit, Snatch is able to realize both direct and indirect window methods by achieving another periodical forwarding with a second time counter registers. The modifications should be made at the compiling phase.

Reduce methods include `count`, `countByValue`, `countByValueAndWindow`, `countByWindow`, `groupByKey`, `groupByKeyAndWindow`, `reduce`, `reduceByKey`, `reduceByKeyAndWindow`, and `reduceByWindow`. Among them, count and groupByKey and their associated methods can be regarded as special cases for reduce and associated methods, and they have been implemented in my Snatch prototype. Reduce and associated methods fit in the match and action programming model, and thus should be supported by INSA as long as the input function is supported by INSA. The modifications should be made at the compiling phase.

Finally, table-join methods include `cogroup`, `join`, `fullOuterJoin`, `leftOuterJoin`, `rightOuterJoin`, and `union`. These methods correspond to SQL join clauses which combine the columns from one or more tables. Snatch's cookie/data-stack has very similar data structure from tables, and technically it is possible to perform the join method at AggSwitch by storing all the cookie/data from periodical aggregation packets (representing DStreams) in the switch and then construct another custom packet as a result of join and deliver it to the analytics server. For instance, I take the `fullOuterJoin` as an example. Stream 1 has cookies A, B, C whereas Stream 2 has cookies A, D, E. AggSwitch reserves a register space for a table with columns A, B, C, D, E. When collecting periodical aggregation packets from LarkSwitches, what AggSwitch needs to do is simply fill in the registers according to the value in cookie A. Thus, when all the periodical aggregation packets are received, AggSwitch has a full table of the result of `fullOuterJoin` on Stream 1 and 2. Other table-join operations can be done in a similar spirit. Here, the modifications should be made at both the compiling phase and the application submission phase.

Note that table-join methods might be beneficial when applying to two separate applications per the developers' agreement, which is a topic I plan to explore in future work. Otherwise, it is a bad practice since it costs too much of the switch storage resources and a better design of the cookie/data-stack will remove the necessity for the join operation.

One limitation is that complex operands, *e.g.,* modulo and logarithm, are not supported by most P4 devices. Nevertheless, this can be resolved by using FPGA-based devices [300], redesigning the algorithms [321], or using P4's `digest` to complete the operations with the help of the control plane [72]. Further, machine learning algorithms or their pre-processing can also be completed in the programmable data plane [274, 316].

To sum up, despite the limitations, programmable switches' ability to process data at a high speed and low power cost is a great asset to boost up the performance of Snatch.

4.3.2 Clients and Servers

I target minimal client modification. For QUIC 0-RTT, the client does not need any modification. For QUIC 1-RTT, a minor change in userspace is needed so that the transport-layer cookies from the last connection are stored and repeated in the next connection, while the rest of the connection ID is randomly re-generated. I implement a Snatch client based on quic-go [60]. I realize the transport-layer cookie support for QUIC 1-RTT by modifying only <50 lines of code. I further implement the Snatch-enabled edge- and web-server, also based on the quic-go repository.

4.3.3 Controller

Functionality. Snatch controller takes inputs from the application developers. It then generates a random byte as the application ID and a random AES-128 key for semantic cookie encryption. Then, it updates the components in the following order: AggSwitch, LarkSwitches, and the

edge servers. With corresponding programs pre-installed at all the rest components, Snatch controller only needs to update the parameters, *e.g.,* altering the table entries in LarkSwitches and AggSwitches so they can recognize new applications and send results to new destinations, through RPCs to the corresponding control plane. The update frequency is overall low, *e.g.,* days or weeks, because updates only happen when new applications are added or AES keys need to be updated.

Consistency. When a controller updates an application, inconsistency issues might arise because of the delay between the controller and other components. For instance, some edge servers might change the format of transport-layer cookies before a LarkSwitch, or a LarkSwitch changes the recognition of the cookie-stack before changes are made. They may result in missing or incorrect results being reported.

I solve the inconsistency issue by adopting a version control scheme. When an update instruction is received by the controller, it generates a new application version with a new application identifier, *i.e.,* the same application has different application IDs for different versions. It then updates the components in order: AggSwitch, LarkSwitches, and the edge servers. After a period of time (possibly days), the controller deletes the old application ID and associated rules, *i.e.,* revokes the corresponding rules on the AggSwitch and LarkSwitches. In this way, Snatch ensures that consistency is preserved when updating the applications.

4.4 Evaluation

In this section, I first present results from my global measurement study on understanding the performance of data streaming from normal Internet users. I then simulate and evaluate the benefits of my approach with my testbed that simulates real-world environments.

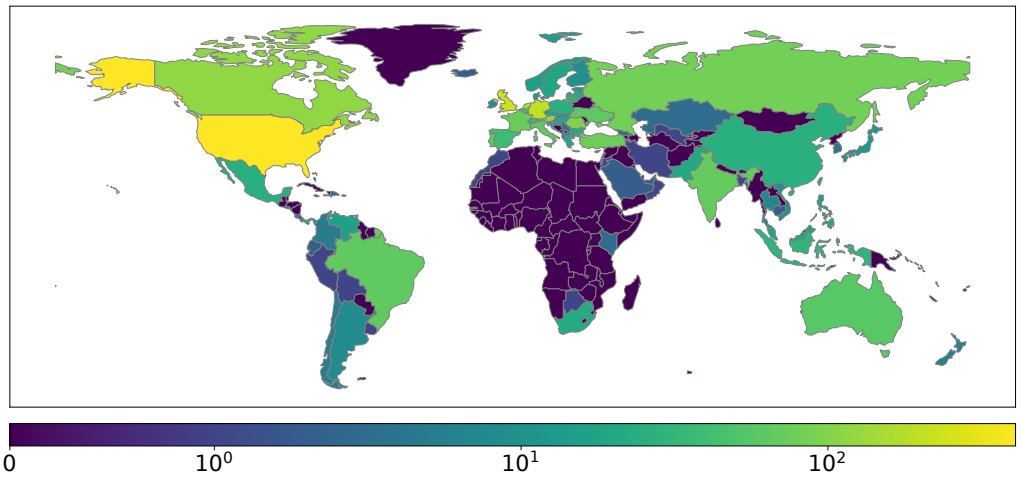


Figure 4.5: Overview of measurement sites.

4.4.1 Measurement and Estimation

Methodology. Snatch involves multiple Internet components: the ISP switch, edge server, web server, and analytics server. To study the performance of these subjects in practice, I set up experiments as follows. First, I host HTTPS websites using AWS EC2 instances [61], which represent the web servers in Figure 4.1(b). Then, I purchase CDN services from Cloudflare [66] and AWS Cloud Front [67]. This allows me to set up the edge servers on a global scale.

Next, I need to measure the performance of regular Internet users on a global scale. I choose the decentralized VPNs (dVPNs), which have gained much popularity recently (see chapter 2), as my means of measurement over academic measurement platforms [142, 161, 272, 288] for better flexibility. In dVPNs, regular Internet users from all over the world monetize their spare bandwidth by hosting a VPN proxy at their homes, providing a VPN service to the public. Thus, it provides a desirable measurement platform for this study. I select Mysterium [21] among various dVPNs since it has the largest footprint (see chapter 2) – at the time of this project, Mysterium currently holds over 5,000 dVPN nodes (proxies), among which over 2,000 are recognized as "residential,"

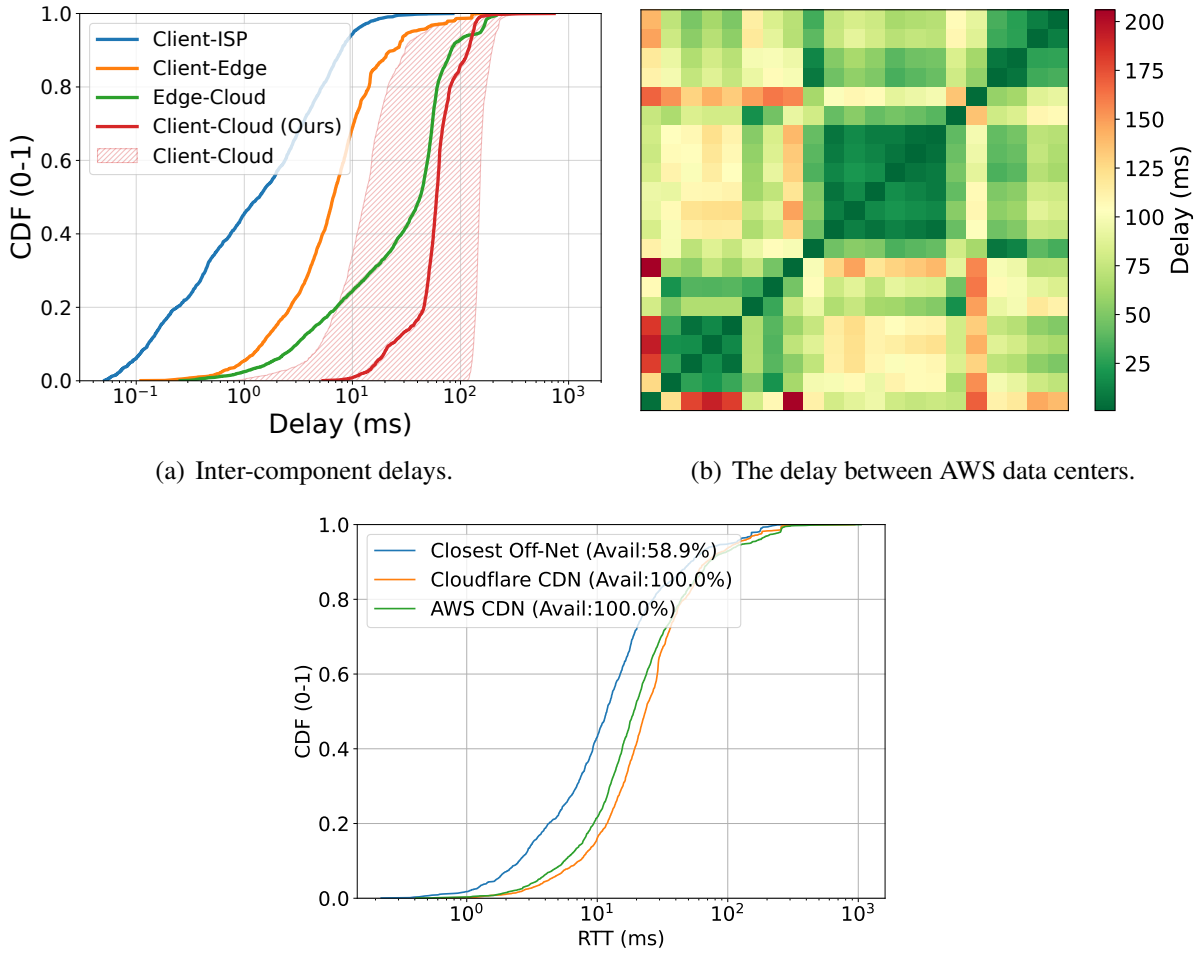
i.e., hosted in regular Internet users' home networks.

I iteratively connect to all the available residential dVPN nodes as measurement sites. During each dVPN connection, I send out various packets and derive delays between Snatch components. For all the per-site operations, I iterate 10 times and take the median for further analysis to avoid outliers resulting from unstable network conditions.

Measurement Results. I conduct my measurement over 14 days, during which I tested 2,253 sites (dVPN nodes) around the world. Figure 4.5 shows the per-country site counts. Among 87 countries I have investigated, the US has the most sites, followed by the UK and Germany. It is noteworthy that while the measured sites are not representative of billions of Internet users, they allow us to capture a glimpse of the current global WAN practice and provide a meaningful basis to estimate the potential benefits of Snatch. Also, the number of sites is not entirely proportional to the total number of Internet users per country, but they represent the user engagement to a large extent. Thus, I utilize such collected statistics to evaluate Snatch.

Figure 4.6(a) shows the delays between client, ISP, edge (server), and cloud (web server and analytics server), respectively. The delay from client to ISP is the smallest as expected, with a median of 1.4 ms. The delay from the client to the edge is slightly larger, with a median of 6.7 ms. This shows the success of CDN services as a means to improve Internet performance, and also Snatch's potentials from the semantic cookie early forwarding. For each site, I take the minimal delay from the off-net servers, Cloudflare CDN, and AWS Cloud Front.

Next, I look at cloud performance. Figure 4.6(a) shows that the delays from client to the cloud (dashed red area) vary a lot – from 13.1 ms to 150.3 ms in the median – depending on the relative geolocation. Further, the median delays from the client and from the edge to my hosted EC2 machines are 60.1 ms and 43.6 ms (red and green lines), respectively. Note that the sum delays from the client to the edge and from the edge to the cloud are not always equal to the delay from



(a) Inter-component delays.

(b) The delay between AWS data centers.

(c) Delays from Internet client to different edge servers.

Figure 4.6: Measurement results.

the client to the cloud, because of the complex routing policies across ASes which may not assign the same path [118].

Moreover, Figure 4.6(b) shows the matrix of intra- and inter-data center delays of the AWS cloud. The delays range from 0.8 ms (within the same data center) to 206 ms (from *ap-southeast-2* region to the *af-south-1* region). The inter-data center median delay is 75.5 ms. The inter-data center delays represent the communication cost from the web server to the analytics server, which

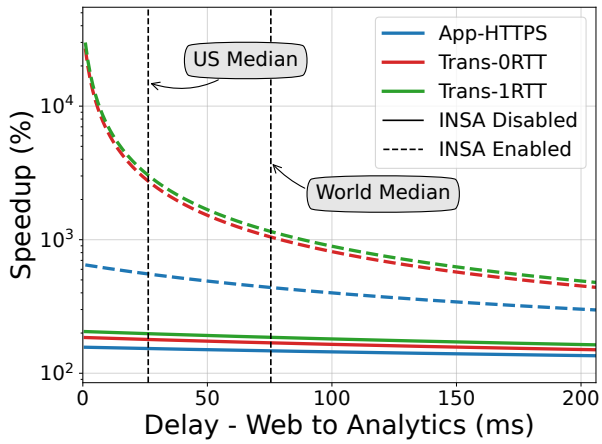
may reside in different data centers as explained in § 4.1.

Last, I comment on the delays from client to different edge servers. Figure 4.6(c) shows that the off-net servers are much closer to the clients compared to regular CDN services, though they cover only 57.9% clients in my measurement. Moreover, Amazon CloudFront outperforms Cloudflare CDN in my measurement. In my analysis in § 4.4, I take the minimal delay among all the edge servers for each client, *i.e.,* if the off-net servers are present and outperform Amazon CloudFront and Cloudflare CDN, then the delay is for the off-net server; otherwise, the delay is the minimum between Amazon CloudFront and Cloudflare CDN.

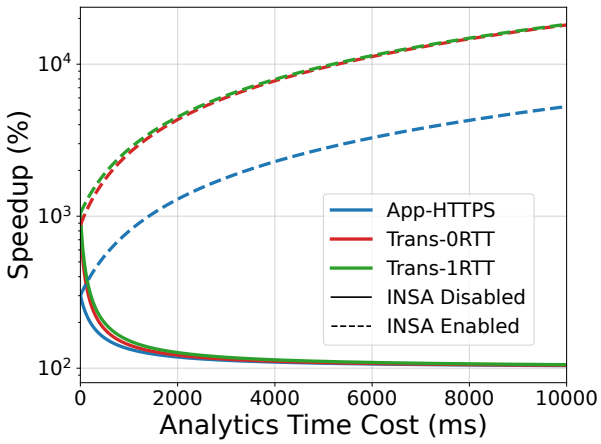
Quantifying Snatch Benefits. Here, I estimate the speedup that Snatch brings. In particular, I utilize the speedup Equations for different protocols, *i.e.,* Equations (4.1), (4.5), and (4.6), combined with the above measurement results. If not otherwise indicated, I estimate based on medians: 1.4 ms for the delay between client and ISP (d_{CI}), 6.7 ms for the delay between client and edge server (d_{CE}), 43.6 ms for the delay between edge and web server (d_{EW}), 0.8 ms for transmission time cost (T_{trans}), 136.6 ms for time cost at the edge (T_E), 241.6 ms for time cost at the web server (T_W), and 500 ms for time cost at the analytics server (T_A) – assuming default Spark parameters [78].

I first investigate the expected Snatch speedup as a function of the median delay between the web server and the analytics server, (d_{WA}). I adopt the "best practice" assumption: the client will always choose the closest edge and web servers. In particular, the delay from the edge to the web server (d_{EW}) is approximated by taking the difference between the delays from the client to the closest cloud and from the client to the edge server, whereas the delay from the edge to the analytics server (d_{EA}) is represented by the "Edge-Cloud" curve in Figure 4.6(a). I further assume that d_{CA} *and* d_{EA} *grow proportionally as the delay* d_{WA}*, within their own range respectively*.

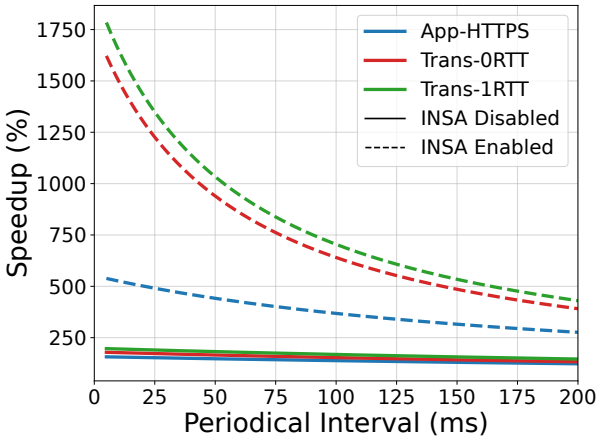
Figure 4.7(a) shows Snatch's speedup as a function of the delay from the web server to the



(a) Speedup of per-packet forwarding as a function of d_{WA}.



(b) Speedup of per-packet forwarding as a function of T_A.



(c) Speedup of periodical forwarding as a function of interval.

Figure 4.7: Speedup estimation.

analytics server (d_{WA}). The solid line represents the case when only early forwarding is enabled ($T'_A = T_A = 500$ ms) whereas the dashed line represents when in-network streaming analytics (INSA for abbreviation) is also enabled ($T'_A = 1$ ms). The figure shows that enabling Snatch's INSA feature improves the performance by a great margin, by up to two orders of magnitude, versus when INSA is disabled. Looking at various protocols, I see that the scenarios where Snatch

benefits the most to least are Trans-1RTT, Trans-0RTT, and APP-HTTPS. This is expected as transport-layer cookies provide better performance than application-layer cookies.

Figure 4.7(a) further shows that as d_{WA} increases, hence d_{CA} and d_{EA} increase following best practice assumption, the Snatch benefits necessarily decrease. Indeed, the more distributed the users are, the network latency more significantly affects Snatch's performance. Next, I focus on two scenarios: (1) US, where end-users are located in the US and the median inter-data-center delay is 26.3 ms, and (2) worldwide, where users are dispersed around the world and the median inter-data-center delay is 75.5 ms (see Figure 4.6(b)). Figure 4.7(a) shows that QUIC 1-RTT INSA speedup is 31x in US and 12x worldwide, while App-HTTPS INSA speedup is 5.5x in US and 4.4x worldwide.

Figure 4.7(b) shows the speedup as a function of analytics time cost, T_A. In practice, the analytics time cost depends on many factors, including the analytics algorithms, workload, the Pub/Sub queuing delays, the settings of traditionally defined analytics systems, *etc*. The time cost thus ranges from negligible to $\sim$10 seconds at the hyper-giants [73]. Here I consider general tasks and hence vary T_A from 1 ms to 10s. When T_A is negligible, INSA naturally does not play an important role. But as T_A grows, the speedups diverge: they decrease when INSA is disabled but increase when INSA is enabled. Overall, Snatch always boosts up the performance of streaming analytics. For reference, when T_A is 10s, and INSA is enabled, the speedup for Trans-1RTT is 183x, for Trans-0RTT is 181x, and for App-HTTPS is 53x.

Figure 4.7(c) shows the speedup in the case of periodical forwarding, as a function of the period (interval) ranging from 5 ms to 200 ms. When the interval is 5 ms, the speedup is naturally closer to per-packet forwarding. As expected, the speedup decreases when the interval increases because the data takes more time to the analytics server. For reference, for interval of 5 ms, the speedup for Trans-1RTT is 18x, while for interval of 200 ms, the speedup for Trans-1RTT is 4.3x.

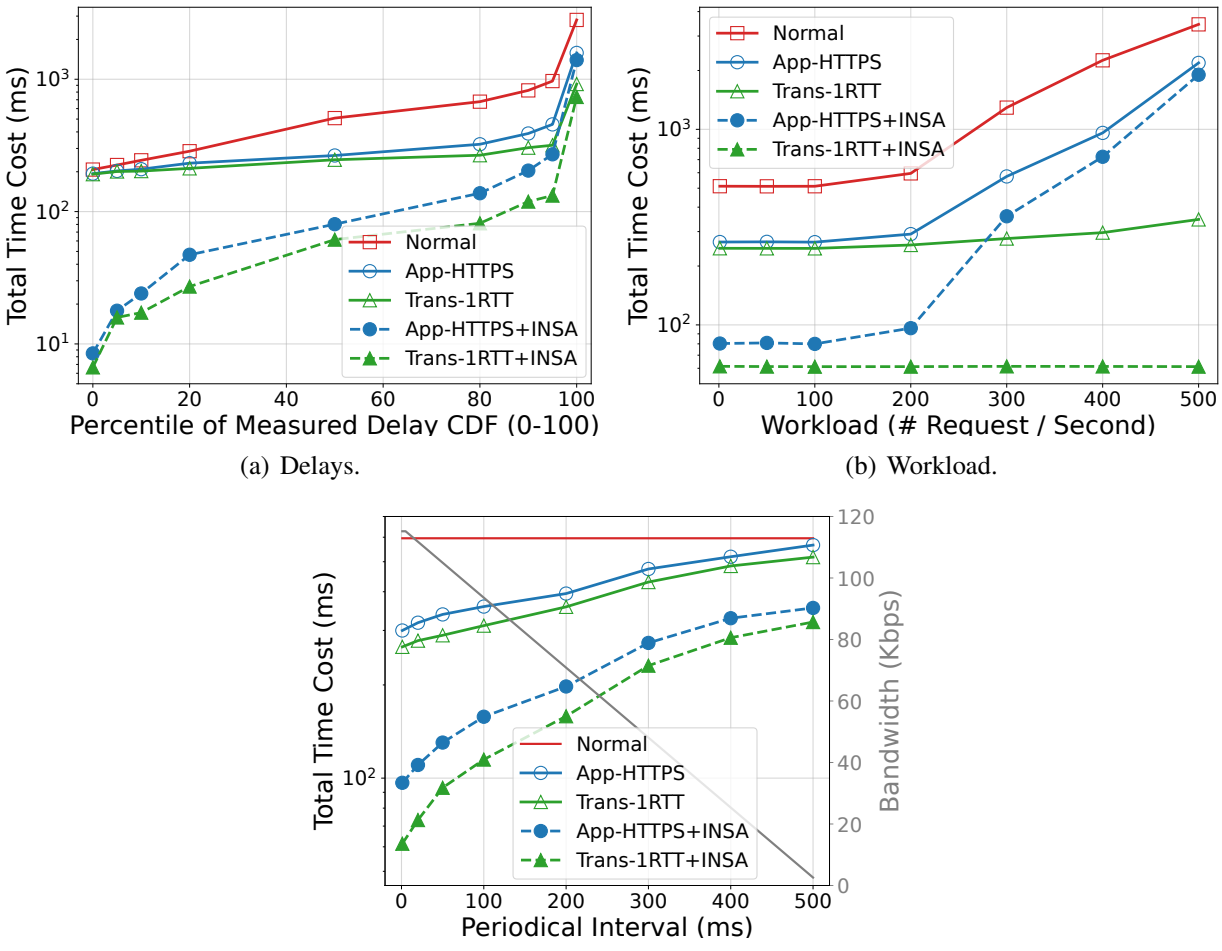
4.4.2 Testbed Experiments

Environment Setup. I set up a testbed consisting of 6 host machines and one Tofino programmable switch. Among them, three host machines represent the client (request generator), the edge server, and the web server, respectively. The analytics server is represented by a cluster of three machines, which consist of two slave nodes and one master node. The Tofino switch represents both Lark-Switch and AggSwitch. The topology follows Figure 4.3 where the Tofino switch connects to the client, the master node of the analytics server, and the edge server (with two different ports connecting to two different network interfaces, respectively). Each host machine is equipped with an 8-core AMD EPYC CPU with 16GB RAM. The delays between the machines are controlled via Linux `Traffic Control` module [36].

I adopt QUIC 1-RTT in my evaluation below. Once the analytics server (master node) receives an aggregation packet, meaning that INSA is enabled, it will log the timestamp; otherwise, it will submit the data to Spark Streaming which processes the data and logs the finishing timestamp.

I select the advertisement campaign analytics as the target application. Different from Yahoo Streaming Benchmark [140], which correlates the user ID and the ad campaign ID, I go further to count the user demographic information (I randomly generate gender, age, and geolocation for each user) for each ad campaign. In addition, I set the interval of Spark Streaming to be 150 ms as it is suitable for most of my tasks and environment, *i.e.*, it minimizes the time cost.

Performance Evaluation. I first evaluate the impact of delays between the components for per-packet forwarding. I adopt different delays—taking Nth percentile of delays from Figure 4.6(a)—in my testbed and perform 10,000 requests from the client for each experiment. I send 10 requests per second (RPS), a relatively low rate, to exclude the impact of workloads (which I explore later in the text). I adopt the same "best practice" assumption as in § 4.4.1.



(a) Delays.

(b) Workload.

(c) Periodical interval.

Figure 4.8: Testbed evaluations. Total time cost as functions of (a) delays, (b) workload, and (c) periodical interval.

Figure 4.8(a) shows the total time costs given different delay percentiles in my measurement. The total time costs are measured from when clients send requests until the results are obtained, either from Spark Streaming (solid lines and hollow markers) or from AggSwitch if INSA is enabled (dashed lines and filled markers). Overall, the results show that the total time cost increases as the delay percentile increases, *i.e.,* the clients experience worse Internet infrastructure. Still, Snatch is beneficial at all times. The shortest to longest total time cost are between Trans-1RTT

and App-HTTPS with INSA, and between Trans-1RTT and App-HTTPS without INSA. One exception occurs at the 100th delay percentile where the performance of Trans-1RTT without INSA exceeds App-HTTPS with INSA because d_{CE} drastically increases.

In terms of speedups, APP-HTTPS and Trans-1RTT reduce the time cost at most by a factor of 2.1x and 5.4x without INSA (at 95th delay percentile), or by 24.5x and 31.2x with INSA (at 1st delay percentile). When INSA is enabled, the speedups slowly decrease as the delay percentile increase, *i.e.,* the clients experience worse Internet infrastructure. Yet, the speedup of Trans-1RTT with INSA is at least 3.8x at the 100th delay percentile, which brings the total time from 2,807 ms down to 735 ms. In the median case, the speedups for APP-HTTPS and Trans-1RTT are 1.9x and 2.0x without INSA, or 6.3x and 8.3x with INSA. Compared to Figure 4.7(a), Trans-1RTT underperforms the results from § 4.4.1, yet App-HTTPS over-performs the corresponding results. This is because the processing time costs at the edge server and at the analytics server in my testbed are both smaller than in § 4.4.1.

Next, I evaluate the impact of the workload. I take the median delays from the measurement, and adjust the workload, which I quantify as the number of requests that the clients send per second. I consider per-packet forwarding here because it consumes more bandwidth and is thus more sensitive to workload compared to periodical forwarding. Figure 4.8(b) shows that the total time costs are stable with the same rank as in Figure 4.8(a) when the workload is relatively low ($<$100). Later, *i.e.,* when workload $>$100, the total time costs increase as the workload increases for all scenarios except Trans-1RTT with INSA, demonstrating the power of in-network transport-layer switch-based processing. When the workload is equal to or greater than 300, the time costs for no-Snatch and App-HTTPS start to increase sharply (note that the y-axis of Figure 4.8(b) is in log scale). Likewise, App-HTTPS with INSA is less effective than Trans-1RTT without INSA. This suggests that congestion happens at the edge server and the web servers because they are

overwhelmed by the high request rate.

Meanwhile, however, Trans-1RTT with INSA keeps a very stable performance – it takes 61 ms regardless of the workload. This reveals a property of Snatch: *no parallelism inflation*. The stable performance is expected because of the nature of line-rate processing of programmable switches and the design of Snatch: Trans-1RTT skips all the computation on the edge and web servers (and the analytics server if INSA is enabled) where congestion may happen at a high workload. In fact, Trans-1RTT and Trans-0RTT are able to keep the best performance as long as the throughput does not exceed the capacity of the switches, which is over 10 Tbps [69].

Finally, I evaluate the periodical forwarding. I adopt the median delays and a workload of 200 RPS. Figure 4.8(c) shows that as the periodical interval increases, the total time cost increases while the bandwidth consumption (grey line) between LarkSwitch/the edge server and AggSwitch decreases. Nevertheless, when the periodical interval is 500 ms, Trans-1RTT and App-HTTPS still speed up the total time cost by 1.2x and 1.1x without INSA, or 1.8x and 1.7x with INSA. The bandwidth consumption linearly decreases from $\sim$112 Kbps to $\sim$1 Kbps as the periodical interval increases from less than 5 ms to 500 ms.

4.5 Related Work

Streaming Analytics. In addition to streaming analytics systems discussed in § 4.1, JetStream [262] and AWStream [323] explore the wide-area streaming analytics whose data sources are widely distributed and propose to reduce the data rate to cope with the limited WAN bandwidth. In addition, Iridium [260] optimizes the data placement before the arrivals of queries. Sana [202] applies WAN-aware multi-query optimization. The wide-area streaming analytics assumes that the data is heading *directly* to the analytics server after it is generated. This is however different from my concerned scenarios where the data accompanies the user requests and thus makes a detour. Snatch

removes this detour and enables in-network analytics via semantic cookies.

In-Network Computation. With the advent of programmable networking hardware and programming languages [110, 111, 286], researchers have proposed to leverage in-network computation to handle network management [224], caching [200], load balancers [242], replicated storage [199], lock management [320], deep neural network training [220, 274], *etc*. Ports *et al.* [259] summarizes a general guide of what and when to offload the computation to the network. While most work targets scenarios within data centers, Jagen targets ISP-centric defense [227]. Snatch aims to speed up online streaming analytics by leveraging the in-network computation and in cooperation with both the ISPs and the cloud.

QUIC Protocol. QUIC [219] has become an official IETF recognized transport-layer protocol in 2021 [196], and has been supported by multiple major browsers including Chrome and Firefox. QUIC is based on UDP and is implemented at userspace rather than kernel space like other transport protocols, which enables it to be more flexible and to adapt to new algorithms. Researchers have started to investigate and leverage the features of QUIC. Connection ID, for instance, is exploited for encoding information, which enables to assist load balancer and hardware accelerators [102, 162]. In this project, I leverage the QUIC Connection ID to encode application-level information and achieve early processing and forwarding for the online streaming analytics.

Anonymity Preservation. The anonymity preservation research spans across different fields including social networks [170, 269], crowd-sourcing [194], recommendations [318], *etc*. One approach is differential privacy, *i.e.,* adding structural noise to its data to report [163], and thus prevent the attackers from inspecting what each user actually sends while ensuring that the aggregated results are statistically correct. It is noteworthy that in differential privacy, there exists a trade-off between too little noise, such that not enough privacy guarantee is provided, and too much noise, such that some data is lost in the noise [166]. Another approach is using secure multi-party com-

putation protocols, where a set of non-colluding servers privately perform computation over the user data [145]. However, MPC methods often come with significant overhead [117, 225, 249]. In general, the common challenges for all privacy-preserving analytics include a high cost and robustness towards malicious users and servers. A third approach is to make the users send data through an anonymizing network, *e.g.,* mix-net [115, 218] or Tor [189, 258], where the data and individual identities are decoupled. However, these methods incur a high cost [146, 306]. My proposal instead prevents the user from sending personally identifiable information by design. To defend against the honest but curious edge, my proposal also leverages differential privacy to protect user information.

4.6 Discussion

Semantic Cookie Related Issues. One question may raise on how the application developers can derive the semantic information without a user ID. In fact, we can regard the semantic cookie as a state machine: the developers have the state from the last request, update it based on the current request, and save it on the users' side for the next request.

Another issue may be the additional overhead from adopting semantic cookies. Transport-layer semantic cookies do not incur any overhead as an existing header field of QUIC with limited length is used. Application-layer semantic cookies inherit the current cookie design but *ideally* only discard the individual identifiers, which brings no overhead. Still, overhead may be introduced by the way that the developers design the application-layer semantic cookies. Currently, the developers build their own database and store as much user information as they want, *e.g.,* the complete visit history per user [295]. With the semantic cookies, the developers can only collect the visit history by appending the new visit to the semantic cookies every time the user visits the website. This will indeed bring non-trivial overhead. Nevertheless, while no hard restriction on the size of se-

mantic cookies is applied, I argue that this is a feature rather than a defect: the semantic cookies are meant to prevent the developers from logging everything about the user, *e.g.,* complete visit history. Hence, it forces the developers to carefully re-design the cookies and only ask for the least; otherwise, they may lose customers because of bad experiences.

Alternative to Latency Inflation. One alternative to reduce latency inflation introduced in § 4.1.3 is to ask the users to send duplicate requests to both the web servers and the analytics servers. Yet, there are many drawbacks from this approach. First and most importantly, it does not enhance user privacy as Snatch does because individual identifiers are still present. Second, it cannot benefit from in-network computation, which may be a larger factor in performance improvement than latency inflation (see § 4.4). Third, it requires the users to double their bandwidth consumption and leads to a worse web experience, yet without offering any incentives to the users. In addition, exposing the analytics server to public may open the door to attacks.

View From Application Developers. With Snatch, application developers can benefit from faster online streaming analytics and hence obtain more valuable results. Meanwhile, they lose the freedom to store whatever they want from the users' activities and may fail to perform certain analytics, *e.g.,* individual profiling [295]. Nevertheless, more studies are looking into how to effectively perform anonymity-preserving analytics [107, 159, 293]. It is thus questionable how much the cost really is from discarding individual-level analytics. Moreover, developers may lose the freedom anyway as stricter privacy policies may be enforced given the public's rising privacy concerns. In addition, the developers can actually benefit from respecting user privacy: users who care about their privacy may be more inclined to websites that adopt semantic cookies compared to other competing websites. This may become an important incentive for more developers to adopt the semantic cookies, and (hopefully) eventually lead to widespread adoption of semantic cookies, similar to the history of HTTPS adoption.

Generality of Analytics. In my implementation, I pre-install programs at the edge devices and have them accept RPCs from Snatch controller to update certain parameters (§ 4.3.3). This would allow edge devices to recognize new applications and perform analytics accordingly. Yet, I acknowledge that my implementation only supports fixed types of aggregation analytics. While the edge servers should be able to conduct any streaming analytics, I have analyzed the capabilities as well as the limitations of the programmable switches (§ 4.3.1). In an ideal implementation, the controller should generate efficient and on-demand codes and push them to the edge devices. I leave this as future work.

Fault Tolerance. Snatch might fail due to various issues. For example, inconsistency might occur when the controller tries to update other components (see § 4.3.3). Other examples include failing to update AES keys at edge servers, or packet drops, *etc*. All these issues will result in the same outcome: the aggregated results become inaccurate. Fortunately, I can detect such failures by running the same analytics on data that is collected from the web servers and arrives at a later time. Application developers should report the result difference to the Snatch controller, which would then check and update the other components through RPCs. I leave the real-time detection and correction for future work.

In-Network Streaming Analytics Trade-offs. In the evaluation, I consider that either INSA is enabled or disabled. In practice, and for most real-world scenarios, the speedup is in between because of the complexity of queries. When more computation is offloaded to the network, the speedup is higher given the negligible time cost for the processing at the switches. Still, more computation also incurs more switch resources, *i.e.,* fewer applications can utilize the switches' support. Thus, there exists a trade-off for the ISPs: support more applications with a smaller speedup for each, or support fewer applications with a larger speedup for each. Independently, Snatch provides a considerable speedup compared to the state-of-the-art even when INSA is disabled.

UDP for the custom aggregation packet. The extracted cookies and data encoded in the custom aggregation packet from LarkSwitch and edge server to AggSwitch may be lost because UDP is used. I argue that the benefits of using UDP overtake the loss. The loss here is that less than 0.01%, *i.e.,* the packet drop rate in today's WAN [70, 71, 77], of the cookies or data will be lost. In comparison, there are two major benefits. First, for short-term analysis, which is the target for Snatch, the value of the data is much higher when the data is available sooner. Dropping one data point out of tens or hundreds of thousands will not make a large difference to the distribution of the data, and thus to the results. At the same time, the data is not lost forever. For a long-term analysis, full and accurate results can be obtained by syncing up the records at the web servers or related databases. Second, implementing a retransmission mechanism on programmable switches is non-trivial and consumes scarce DRAM resources to keep the status. Instead, the resources can be used to offload more computation and thus provide better speedup or support more applications. In conclusion, it is the best choice for Snatch to adopt UDP for the custom aggregation packet.

Ethical Consideration. My measurement in § 4.4.1 involves sending requests through proxies located at Internet users' home networks. However, these Internet users are selling their Internet access, and the dVPN service is publicly available. Therefore, this is no different than connecting to traditional VPNs. Further, I did not send any malicious requests or had any operations which might endanger the proxies. Thus, this work does not have any ethical concerns.

4.7 Summary

This chapter presented Snatch, a system that early forwards and pre-processes the online streaming data at the network edge to speed up the online streaming analytics and preserve user anonymity. The key to enabling Snatch is the introduction of semantic cookies, which carry encrypted user information that is personally unidentifiable and directly available for analytics. I demonstrated

that it is viable to encode semantic cookies in the existing application or transport protocols. My evaluation of Snatch – based on real-world measurements – showed that when processing can be done early in-network, Snatch can speed up user analytics by 10-30x. Given the growing trend of migrating infrastructure towards the edge, such speedups along with privacy enhancements are likely to soon become a reality.

CHAPTER 5
CONCLUSION

Given the interconnected, multi-layered, and continuously evolving nature of the Internet, ensuring security and preserving user privacy demand comprehensive, cross-layer, and adaptive improvements across all facets of Internet components. This thesis undertakes a comprehensive exploration of network security and privacy, with a concentrated effort on improving key components of the Internet: proxy-based Internet access, Domain Name Service (DNS), and HTTP for Web traffic.

- The first contribution of this thesis is a comprehensive study of the decentralized VPN (DVPN) ecosystem, actively and passively monitoring major DVPNs – Mysterium, Sentinel, and Tachyon – over six months. This extensive data collection and analysis provide invaluable insights into the footprint, performance, income opportunities, and traffic characteristics of these networks.

  Through passive measurement, I have effectively highlighted several security and privacy issues inherent to DVPNs. Notably, the lack of protection for clients' traffic makes it susceptible to monitoring and interception by DVPN nodes, especially for plaintext packets like (current) unencrypted DNS queries and responses. This vulnerability, even when HTTPS traffic is adopted, underscores the potential for privacy breaches.

  Further, DVPNs, despite their decentralized nature, are not entirely free from centralization issues, particularly with the reliance on centralized endpoints for service initiation. This centralization presents a significant risk of attack and undermines the censorship resistance that many DVPN clients seek.

- The second contribution of this thesis is the development of PDNS, which introduces a novel approach to DNS queries that significantly enhances user privacy. PDNS revises a key system premise of DNS by altering the role of recursive resolvers. More concretely, by leveraging Private Information Retrieval (PIR) techniques, PDNS allows recursive resolvers to operate in a manner where they can resolve domain names without actually knowing what those domains are. This design minimizes privacy risks associated with DNS queries, representing a fundamental shift from traditional DNS query mechanisms.

  Nevertheless, incorporating PIR into DNS is not straightforward; as DNS has a hierarchical structure and the DNS resolvers, where PIR is supposed to be implemented, need to refresh their cache by contacting authoritative name servers. To mitigate this issue, PDNS design requires users to undertake iterative DNS lookups directly to authoritative name servers in the event of a cache miss. This approach ensures the resolver's cache is updated without revealing user identity at the resolver. Recognizing the performance challenges associated with implementing PIR in DNS, the thesis presents various optimizations to make PDNS viable while minimizing privacy exposure in the communication between users and authoritative name servers.

  From a practical perspective, the thesis outlines a strategic approach for the gradual deployment of PDNS, highlighting its compatibility and coexistence with the existing DNS infrastructure. This pragmatic approach to deployment emphasizes PDNS's role as an augmentative technology, rather than a replacement, facilitating its adoption alongside current DNS technologies like DoH and ODoH. Through comprehensive benchmarking and analysis, the thesis provides evidence of PDNS's practicality and efficiency. It details the system's performance in real-world scenarios, including its computational and memory requirements, thereby validating the proposed PDNS design's feasibility and effectiveness in enhancing

DNS privacy and security.

- The third contribution of this thesis is the introduction of semantic cookies, designed to address privacy breaches inherent in contemporary HTTP cookie practices, alongside the development of the Snatch system for comprehensive system component coordination. Semantic cookies diverge from traditional practices by embedding non-identifiable user attributes directly within the cookies. This system premise revision enables preliminary data processing and analytics at the network's edge, eliminating the necessity of backend databases behind the scenes and significantly enhancing user privacy protection.

  The use of semantic cookies also transforms the landscape of online streaming analytics by enabling in-network processing. This shift not only improves the efficiency and speed of data analysis but also opens new possibilities for real-time insights and decision-making based on user activities and preferences.

  This thesis has demonstrated that semantic cookies can be deployed without altering existing Web protocols, offering a seamless upgrade path for enhancing privacy and analytics capabilities on the Internet. This ease of deployment encourages adoption and integration into the current Web.

Overall, this thesis underscores the critical importance of re-evaluating existing Internet components. By adopting a holistic approach that considers different Internet applications and services, and by revising system premises, this work presents a sample blueprint for future innovations in building resilient and efficient network systems for enhanced privacy and security.

5.1 Limitations and Future Work

This thesis lays the groundwork, rather than serving as the definitive conclusion, in the evolving quest to fortify network security and privacy. Recognizing the limitations of the projects discussed herein, it underscores the imperative for continuous development towards more resilient systems that elevate the standards of Internet security and privacy.

The first project conducts an in-depth examination of vulnerabilities in current DVPNs, highlighting the critical need to minimize trust reliance on proxies. It thus leaves a goal for future research: to address the significant challenges associated with diminishing trust in proxies, thereby realizing a fully secure and private proxy-based architecture

In the second project, the PDNS prototype showcases significant promise in enhancing the privacy protections of contemporary DNS systems. However, there remains considerable scope for optimizing its performance. Additionally, the real-world deployment of PDNS necessitates the integration of numerous DNS features that have been developed over the past decades.

The third project of this thesis highlights the crucial role of differential privacy as a mechanism to safeguard against unauthorized access and analysis by third parties or malicious entities. Yet, the refinement of the differential privacy algorithm demands further development, alongside a detailed evaluation of the data and analysis needs of the Web providers.

Further, while this thesis proposes enhancements to several core Web components, it acknowledges a limitation in the uniformity of solutions across different systems. For instance, semantic cookies and DVPN offer lower privacy assurances compared to PDNS, as they necessitate a degree of trust in various system entities. This disparity stems partly from the thesis's pragmatic philosophy, which prioritizes practicality. However, further advancements are essential to bolster the privacy protections of these components, aiming for a more cohesive alignment of privacy

guarantees.

Moreover, this thesis focuses on technical enhancements to Web components. It is important to acknowledge that policy significantly influences this area as well. Future research should also explore the development of systems that adhere to policies, perform thorough reviews of policy practices, and engage in discussions about policy directions.

Finally, the Internet encompasses a vast array of components beyond those covered in this thesis, such as advertising systems, search engines, and more. These domains represent crucial opportunities for ongoing research and innovation, essential for the continued evolution of the Internet and to address the dynamic needs of its users.

REFERENCES

[1] Add Service - AWS Pricing Calculator. https://calculator.aws/#/addService/ec2-enhancement. Accessed in 2023.

[2] Aws route 53. https://aws.amazon.com/route53/.

[3] Bind 9. https://www.isc.org/bind/.

[4] BIND DoH Update. https://www.isc.org/blogs/bind-doh-update-2021/.

[5] Blyss (Previously Spiral-rs). https://github.com/blyssprivacy/sdk.

[6] Customer URL Ticketing System. https://www.trustedsource.org/sources/index.pl.

[7] dig(1): DNS lookup utility - Linux man page - Die.net. https://linux.die.net/man/1/dig.

[8] Dnscrypt. https://www.dnscrypt.org/.

[9] Google CDN ip-range. https://groups.google.com/g/gce-discussion/c/V2n9Ri-T5qg.

[10] Google Cloud Pricing Calculator. https://calculator.aws/#/addService/ec2-enhancement. Accessed in 2023.

[11] Hola: Get access to worldwide content. https://hola.org/.

[12] HTTPS encryption on the web. https://transparencyreport.google.com/https/overview.

[13] Intel® Software Guard Extensions (Intel® SGX) SDK for Linux* OS Developer Reference. https://download.01.org/intel-sgx/sgx-linux/2.10/docs/Intel_SGX_Developer_Reference_Linux_2.10_Open_Source.pdf. Accessed in 2023.

[14] Internet Deals, Plans, and Pricing | Xfinity. https://www.xfinity.com/learn/internet-service/deals.

[15] Intrinsics for Intel® Advanced Vector Extensions 2 (Intel® AVX2). https://www.intel.com/content/www/us/en/develop/documentation/cpp-compiler-developer-guide-and-reference/top/compiler-reference/intrinsics/intrinsics-for-avx2.html.

[16] Key Features of Tachyon Protocol. https://tachyon.eco/?n=yr8mtzfwee.WhatIsTachyon.

[17] Lethean: Absolute Internet Privacy. https://lethean.io/.

[18] Linux traffic control (tc). https://man7.org/linux/man-pages/man8/tc.8.html.

[19] Locations and IP address ranges of CloudFront edge servers. https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/LocationsOfEdgeServers.html.

[20] MaxMind: IP Geolocation and Online Fraud Prevention. https://www.maxmind.com/.

[21] Mysterium network: Censorship free Internet for all. https://mysterium.network/.

[22] Mysterium Network, Sentinel Launch dVPN Alliance To Make Internet Users "Untraceable, Unblockable And Unhackable" . https://www.globenewswire.com/news-release/2020/10/14/2108293/0/en/Mysterium-Network-Sentinel-Launch-dVPN-Alliance-To-Make-Internet-Users-Untraceable-Unblockable-And-Unhackable.html. Accessed in October 2020.

[23] OpenVPN. https://openvpn.net/.

[24] Orchid: The Crypto Powered VPN. https://www.orchid.com/.

[25] Privatix: Next-gen VPN. https://privatix.com/.

[26] ProtonVPN. https://protonvpn.com/.

[27] Reference Guide - McAfee TrustedSource Web Database. https://www.trustedsource.org/download/ts_wd_reference_guide.pdf.

[28] Retrieve the current POP IP list for Azure CDN. https://docs.microsoft.com/en-us/azure/cdn/cdn-pop-list-api.

[29] RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3. http://tools.ietf.org/html/rfc8446.

[30] RING. https://github.com/yunmingxiao/RING. Accessed in June 2022.

[31] RING: the first multi-vendors bandwidth marketplace. https://ringdvpn.com/. Accessed in June 2022.

[32] Sentinel: Secure yourself today with Sentinel. sentinel.co.

[33] Speedtest Global Index. https://www.speedtest.net/global-index.

[34] Speedtest Global Index - United States. https://www.speedtest.net/global-index/united-states.

[35] Tachyon VPN. https://tachyon.eco/.

[36] tc(8) — Linux manual page. https://man7.org/linux/man-pages/man8/tc.8.html.

[37] Testnet 1.0 ends as Testnet 2.0 unlocks new milestone. https://mysterium.network/blog/testnet-1-0-ends-as-testnet-2-0-unlocks-new-milestone/.

[38] USA v. Sussmann, 2022 WL 1124755 (D.D.C. 2022).

[39] The Truth About Faster Internet: It's Not Worth It, The Wall Street Journal, 2019. https://www.wsj.com/graphics/faster-internet-not-worth-it/.

[40] Tor Project | Anonymity Online. https://www.torproject.org/.

[41] Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. https://tranco-list.eu/.

[42] Tstat - TCP STatistic and Analysis Tool. http://tstat.polito.it/.

[43] VPN Gate - Public VPN Relay Servers. https://www.vpngate.net/en/.

[44] WebPageTest. https://github.com/WPO-Foundation/webpagetest.

[45] What is DNS? | How DNS works | Cloudflare. https://www.cloudflare.com/learning/dns/what-is-dns/.

[46] When to replace BIND DNS. https://bluecatnetworks.com/blog/when-to-replace-bind-dns/.

[47] WireGuard: fast, modern, secure VPN tunnel. https://www.wireguard.com/.

[48] Xfinity Data Plans. https://www.xfinity.com/learn/internet-service/data.

[49] Domain names - implementation and specification. RFC 1035, Nov. 1987.

[50] Alibaba Cloud MMO Gaming Solution Architecture, 2018. https://www.alibabacloud.com/blog/alibaba-cloud-mmo-gaming-solution-architecture_593877.

[51] General Data Protection Regulation (GDPR), 2018. https://gdpr-info.eu/.

[52] Android Debug Bridge (ADB), 2020. https://developer.android.com/studio/command-line/adb/.

[53] Chromium Blog: Building a more private web: A path towards making third party cookies obsolete, 2020. https://blog.chromium.org/2020/01/building-more-private-web-path-towards.html.

[54] Docker, 2020. https://www.docker.com/.

[55] Full Third-Party Cookie Blocking and More, 2020. https://webkit.org/blog/10218/full-third-party-cookie-blocking-and-more/.

[56] Google Safe Browsing, 2020. https://safebrowsing.google.com/.

[57] How Do I Manage Cookies In Brave?, 2020. https://support.brave.com/hc/en-us/articles/360050634931-How-Do-I-Manage-Cookies-In-Brave-.

[58] mitmproxy, 2020. https://mitmproxy.org/.

[59] Privatix Community Update, 2020. https://medium.com/privatix/community-update-c98df60f2c98/.

[60] A QUIC implementation in pure Go, 2021. https://github.com/lucas-clemente/quic-go.

[61] Amazon EC2, 2021. https://aws.amazon.com/ec2/.

[62] Apache Flink – Stateful Computations over Data Streams, 2021. https://flink.apache.org/.

[63] Apache Flume, 2021. https://flume.apache.org/.

[64] Apache Kafka, 2021. https://kafka.apache.org/.

[65] Apache Spark, 2021. https://spark.apache.org/.

[66] Cloudflare – The Web Performance & Security Company, 2021. https://www.cloudflare.com/.

[67] Content Delivery Network (CDN) - Amazon CloudFront, 2021. https://aws.amazon.com/cloudfront/.

[68] Google Cloud Dataflow, 2021. https://cloud.google.com/dataflow.

[69] Intel Tofino 2, 2021. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html.

[70] IP SLA Network Performance – Arelion, 2021. https://www.arelion.com/our-network/ip-sla-network-performance.html.

[71] Our Global IP Network – NTT-GIN, 2021. https://www.arelion.com/our-network/ip-sla-network-performance.html.

[72] P416 Portable Switch Architecture (PSA) – Packet Digest, 2021. https://p4.org/p4-spec/docs/PSA.html#sec-packet-digest.

[73] Processing billions of events in real time at Twitter, 2021. https://blog.twitter.com/engineering/en_us/topics/infrastructure/2021/processing-billions-of-events-in-real-time-at-twitter-.

[74] pyspark.streaming.DStream – PySpark 3.2.0 Documentation, 2021. https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.streaming.DStream.html.

[75] RabbitMQ, 2021. https://www.rabbitmq.com/.

[76] Sentinel api. 2021. https://api.sentinelgroup.io/client/vpn/list.

[77] SLA Performance For Global IP – Sprint, 2021. https://www.sprint.net/tools/sla-performance/sl.

[78] Spark Streaming Programming Guide, 2021. https://spark.apache.org/docs/latest/streaming-programming-guide.html.

[79] Cloudflare DNS, 2022. https://www.cloudflare.com/dns/.

[80] Firefox DNS-over-HTTPS., 2022. https://support.mozilla.org/en-US/kb/firefox-dns-over-https.

[81] Google Public DNS: DNS over HTTPS (DoH), 2022. https://developers.google.com/speed/public-dns/docs/doh.

[82] Introduction to Google Public DNS, 2022. https://developers.google.com/speed/public-dns/docs/intro.

[83] Mira – Real-Time Crowd Analysis, 2022. https://mira.co/solutions/real-time/.

[84] The chromium projects: DNS over HTTPS (aka DoH), 2022. https://www.chromium.org/developers/dns-over-https/.

[85] Firefox rolls out Total Cookie Protection by default to more users worldwide, 2023. https://blog.mozilla.org/en/mozilla/firefox-rolls-out-total-cookie-protection-by-default-to-all-users-worldwide/.

[86] Intel patches up SGX best it can after another load of security holes found, 2023. https://www.theregister.com/2023/02/15/intel_sgx_vulns/.

[87] A. Abhashkumar, J. Lee, J. Tourrilhes, S. Banerjee, W. Wu, J.-M. Kang, and A. Akella. P5: Policy-driven optimization of p4 pipeline. In *Proceedings of the Symposium on SDN Research*, pages 136–142, 2017.

[88] J. Abley and K. E. Lindqvist. Operation of anycast services. *RFC*, 4786:1–24, 2006.

[89] S. Abramova, P. Schöttle, and R. Böhme. Mixing coins of different quality: A game-theoretic approach. In *Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers*, volume 10323 of *Lecture Notes in Computer Science*, pages 280–297. Springer, 2017.

[90] G. Aggarwal, E. Bursztein, C. Jackson, and D. Boneh. An analysis of private browsing modes in modern browsers. In *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*, pages 79–94. USENIX Association, 2010.

[91] T. Akidau, A. Balikov, K. Bekiroğlu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle. Millwheel: Fault-tolerant stream processing at internet scale. *Proceedings of the VLDB Endowment*, 6(11):1033–1044, 2013.

[92] A. Ali, T. Lepoint, S. Patel, M. Raykova, P. Schoppmann, K. Seth, and K. Yeo. {Communication–Computation} trade-offs in {PIR}. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1811–1828, 2021.

[93] M. AlSabah and I. Goldberg. Performance and security improvements for tor: A survey. *ACM Comput. Surv.*, 49(2):32:1–32:36, 2016.

[94] M. Anagnostopoulos, G. Kambourakis, P. Kopanos, G. Louloudakis, and S. Gritzalis. DNS amplification attack revisited. *Comput. Secur.*, 39:475–485, 2013.

[95] S. Angel, H. Chen, K. Laine, and S. Setty. Pir with compressed queries and amortized query processing. In *2018 IEEE symposium on security and privacy (SP)*, pages 962–979. IEEE, 2018.

[96] A. M. Antonopoulos and G. Wood. *Mastering ethereum: building smart contracts and dapps*. O'reilly Media, 2018.

[97] N. J. Apthorpe, D. Y. Huang, D. Reisman, A. Narayanan, and N. Feamster. Keeping the smart home private with smart(er) iot traffic shaping. *Proc. Priv. Enhancing Technol.*, 2019(3):128–148, 2019.

[98] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS security introduction and requirements. *RFC*, 4033:1–21, 2005.

[99] M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, A. Ghodsi, I. Stoica, and M. Zaharia. Structured streaming: A declarative api for real-time applications in apache spark. In *Proceedings of the 2018 International Conference on Management of Data*, pages 601–613, 2018.

[100] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker. Fault-tolerance in the borealis distributed stream processing system. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 13–24, 2005.

[101] H. Ballani and P. Francis. Mitigating DNS dos attacks. In *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pages 189–198. ACM, 2008.

[102] T. Barbette, C. Tang, H. Yao, D. Kostić, G. Q. Maguire Jr, P. Papadimitratos, and M. Chiesa. A high-speed load-balancer design with guaranteed per-connection-consistency. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 667–683, 2020.

[103] E. B. Barker. Secure hash standard (shs). 1995.

[104] A. Bhattacharya, C. Zhan, A. Maji, H. Gupta, S. R. Das, and P. M. Djurić. Selection of sensors for efficient transmitter localization. *IEEE/ACM Transactions on Networking*, 30(1):107–119, 2021.

[105] P. Bischoff. What is a multi-hop VPN and do you need one? `https://www.comparitech.com/blog/vpn-privacy/multi-hop-vpn/`. Accessed in January 2023.

[106] G. Bissias, B. N. Levine, and D. Thibodeau. Using economic risk to model miner hash rate allocation in cryptocurrencies. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2018 International Workshops, DPM 2018 and CBT 2018, Barcelona, Spain, September 6-7, 2018, Proceedings*, volume 11025 of *Lecture Notes in Computer Science*, pages 155–172. Springer, 2018.

[107] A. Bittau, Ú. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnés, and B. Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 441–459. ACM, 2017.

[108] K. Bock, G. Hughey, L.-H. Merino, T. Arya, D. Liscinsky, R. Pogosian, and D. Levin. Come as you are: Helping unmodified clients bypass censorship with server-side evasion. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 586–598, 2020.

[109] S. Bortzmeyer, R. Dolmans, and P. Hoffman. DNS query name minimisation to improve privacy. *RFC*, 9156:1–11, 2021.

[110] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[111] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. *ACM SIGCOMM Computer Communication Review*, 43(4):99–110, 2013.

[112] T. Böttger, F. Cuadrado, G. Antichi, E. L. Fernandes, G. Tyson, I. Castro, and S. Uhlig. An empirical study of the cost of dns-over-https. In *Proceedings of the Internet Measurement Conference*, pages 15–21, 2019.

[113] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[114] A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral. Deep packet inspection as a service. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, CoNEXT 2014, Sydney, Australia, December 2-5, 2014*, pages 271–282. ACM, 2014.

[115] J. Brickell and V. Shmatikov. Efficient anonymity-preserving data collection. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 76–85, 2006.

[116] F. Bronzino, P. Schmitt, S. Ayoubi, G. Martins, R. Teixeira, and N. Feamster. Inferring streaming video quality from encrypted traffic: Practical models and deployment experience. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3):1–25, 2019.

[117] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. {SEPIA}:{Privacy-Preserving} aggregation of {Multi-Domain} network events and statistics. In *19th USENIX Security Symposium (USENIX Security 10)*, 2010.

[118] M. Caesar and J. Rexford. Bgp routing policies in isp networks. *IEEE network*, 19(6):5–11, 2005.

[119] T. Callahan, M. Allman, and M. Rabinovich. On modern DNS behavior and properties. *Comput. Commun. Rev.*, 43(3):7–15, 2013.

[120] P. Callejo, M. Bagnulo, J. G. Ruiz, A. Lutu, A. García-Martínez, and R. Cuevas. Measuring doh with web ads. *Comput. Networks*, 2022.

[121] H. Cao and M. Wachowicz. Analytics everywhere for streaming iot data. In *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pages 18–25. IEEE, 2019.

[122] H. Cao and M. Wachowicz. An edge-fog-cloud architecture of streaming analytics for internet of things applications. *Sensors*, 19(16):3594, 2019.

[123] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, N. Tatbul, S. Zdonik, and M. Stonebraker. Monitoring streams—a new class of data management applications. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 215–226. Elsevier, 2002.

[124] P. Casas, R. Schatz, F. Wamser, M. Seufert, and R. Irmer. Exploring qoe in cellular networks: How much bandwidth do you need for popular smartphone apps? In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, pages 13–18, 2015.

[125] I. Castro, A. Panda, B. Raghavan, S. Shenker, and S. Gorinsky. Route bazaar: Automatic interdomain contract negotiation. In *15th Workshop on Hot Topics in Operating Systems (HotOS {XV})*, 2015.

[126] L. Cerno and T. Amaral. Demand for internet access and use in spain, 2005. `https://eprints.ucm.es/id/eprint/7897/1/0506.pdf`.

[127] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. Telegraphcq: continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 668–668, 2003.

[128] G. J. Chen, J. L. Wiener, S. Iyer, A. Jaiswal, R. Lei, N. Simha, W. Wang, K. Wilfong, T. Williamson, and S. Yilmaz. Realtime data processing at facebook. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1087–1098, 2016.

[129] J. Chen, T. Lan, and N. Choi. Distributional-utility actor-critic for network slice performance guarantee. In *Proceedings of the Twenty-fourth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, pages 161–170, 2023.

[130] J. Chen, T. Lan, and C. Joe-Wong. Rgmcomm: Return gap minimization via discrete communications in multi-agent reinforcement learning, 2023.

[131] J. Chen, Y. Wang, and T. Lan. Bringing fairness to actor-critic reinforcement learning for network utility optimization. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[132] J. Chen, L. Zhang, J. Riem, G. Adam, N. D. Bastian, and T. Lan. Explainable learning-based intrusion detection supported by memristors. In *2023 IEEE Conference on Artificial Intelligence (CAI)*, pages 195–196. IEEE, 2023.

[133] J. Chen, L. Zhang, J. Riem, G. Adam, N. D. Bastian, and T. Lan. Ride: Real-time intrusion detection via explainable machine learning implemented in a memristor hardware architecture. In *2023 IEEE Conference on Dependable and Secure Computing (DSC)*, pages 1–8. IEEE, 2023.

[134] J. Chen, H. Zhou, Y. Mei, G. Adam, N. D. Bastian, and T. Lan. Real-time network intrusion detection via decision transformers. *arXiv preprint arXiv:2312.07696*, 2023.

[135] Q. Chen, P. Ilia, M. Polychronakis, and A. Kapravelos. Cookie swap party: Abusing first-party cookies for web tracking. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 2117–2129. ACM / IW3C2, 2021.

[136] X. Chen. Implementing AES encryption on programmable switches via scrambled lookup tables. In *Proceedings of the 2020 ACM SIGCOMM 2020 Workshop on Secure Programmable Network Infrastructure, SPIN@SIGCOMM 2020, Virtual Event, USA, August 14, 2020*, pages 8–14. ACM, 2020.

[137] X. Chen, C. Liu, B. Li, K. Lu, and D. Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[138] M. Chetty, R. Banks, A. Brush, J. Donner, and R. Grinter. You're capped: understanding the effects of bandwidth caps on broadband use in the home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3021–3030, 2012.

[139] R. Chhabra, P. Murley, D. Kumar, M. Bailey, and G. Wang. Measuring dns-over-https performance around the world. In *IMC '21: ACM Internet Measurement Conference, Virtual Event, USA, November 2-4, 2021*. ACM, 2021.

[140] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, et al. Benchmarking streaming computation engines: Storm, flink and spark streaming. In *2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, pages 1789–1792. IEEE, 2016.

[141] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[142] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[143] C. Contavalli, W. van der Gaast, D. C. Lawrence, and W. Kumari. Client subnet in DNS queries. *RFC*, 7871:1–30, 2016.

[144] S. Corbet, C. Larkin, B. Lucey, A. Meegan, and L. Yarovaya. Cryptocurrency reaction to fomc announcements: Evidence of heterogeneity based on blockchain stack position. *Journal of Financial Stability*, 46:100706, 2020.

[145] H. Corrigan-Gibbs and D. Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 259–282. USENIX Association, 2017.

[146] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE, 2015.

[147] H. Corrigan-Gibbs, A. Henzinger, and D. Kogan. Single-server private information retrieval with sublinear amortized time. *Cryptology ePrint Archive*, 2022.

[148] H. Corrigan-Gibbs and D. Kogan. Private information retrieval with sublinear online time. In *Advances in Cryptology – EUROCRYPT 2020*, pages 44–75, Cham, 2020. Springer International Publishing.

[149] L. Csikor, H. Singh, M. S. Kang, and D. M. Divakaran. Privacy of dns-over-https: Requiem for a dream? In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 252–271. IEEE, 2021.

[150] J. Damas, M. Graff, and P. Vixie. Extension mechanisms for DNS (EDNS(0)). *RFC*, 6891:1–16, 2013.

[151] I. T. U. I. W. T. I. Database. Individuals using the Internet (% of population). `https://data.worldbank.org/indicator/IT.NET.USER.ZS`. Accessed in January 2023.

[152] E. Dauterman and H. Corrigan-Gibbs. Lightweb: Private web browsing without all the baggage. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks, HotNets 2023, Cambridge, MA, USA, November 28-29, 2023*, pages 287–294. ACM, 2023.

[153] M. D. de Assuncao, A. da Silva Veith, and R. Buyya. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. *Journal of Network and Computer Applications*, 103:1–17, 2018.

[154] W. B. de Vries, Q. Scheitle, M. Müller, W. Toorop, R. Dolmans, and R. van Rijswijk-Deij. A first look at QNAME minimization in the domain name system. In *Passive and Active Measurement - 20th International Conference, PAM 2019, Puerto Varas, Chile, March 27-29, 2019, Proceedings*, volume 11419 of *Lecture Notes in Computer Science*, pages 147–160. Springer, 2019.

[155] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[156] G. Di Bella, C. Barcellona, and I. Tinnirello. A secret sharing scheme for anonymous dns queries. In *AEIT Annual Conference 2013*, pages 1–5. IEEE, 2013.

[157] J. Dickinson, S. Dickinson, R. Bellis, A. Mankin, and D. Wessels. DNS Transport over TCP - Implementation Requirements. RFC 7766, Mar. 2016.

[158] T. Dierks and C. Allen. The TLS protocol version 1.0. *RFC*, 2246:1–80, 1999.

[159] B. Ding, J. Kulkarni, and S. Yekhanin. Collecting telemetry data privately. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3571–3580, 2017.

[160] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[161] C. Dovrolis, K. Gummadi, A. Kuzmanovic, and S. D. Meinrath. Measurement lab: Overview and an invitation to the research community, 2010.

[162] M. Duke and N. Banks. QUIC-LB: Generating Routable QUIC Connection IDs. Internet-Draft draft-ietf-quic-load-balancers-10, Internet Engineering Task Force, Jan. 2022. Work in Progress.

[163] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006.

[164] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[165] Ú. Erlingsson, V. Feldman, I. Mironov, A. Raghunathan, K. Talwar, and A. Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2468–2479. SIAM, 2019.

[166] Ú. Erlingsson, V. Pihur, and A. Korolova. RAPPOR: randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1054–1067. ACM, 2014.

[167] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. `https://eprint.iacr.org/2012/144`.

[168] S. Farrell and H. Tschofenig. Pervasive monitoring is an attack. *RFC*, 7258:1–6, 2014.

[169] P. Ferguson and G. Huston. What is a vpn? 1998.

[170] K. B. Frikken and P. Golle. Private social network analysis: how to assemble pieces of a graph privately. In *Proceedings of the 2006 ACM Workshop on Privacy in the Electronic Society, WPES 2006, Alexandria, VA, USA, October 30, 2006*, pages 89–98. ACM, 2006.

[171] K. Fujiwara, A. Sato, and K. Yoshida. Cache effect of shared DNS resolver. *IEICE Trans. Commun.*, 102-B(6):1170–1179, 2019.

[172] C. Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford, CA, USA, 2009. AAI3382729.

[173] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology – CRYPTO 2013*, pages 75–92, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[174] Z. Georgiou, M. Symeonides, D. Trihinas, G. Pallis, and M. D. Dikaiakos. Streamsight: A query-driven framework for streaming analytics in edge computing. In *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, pages 143–152. IEEE, 2018.

[175] P. Gigis, M. Calder, L. Manassakis, G. Nomikos, V. Kotronis, X. Dimitropoulos, E. Katz-Bassett, and G. Smaragdakis. Seven years in the life of hypergiants' off-nets. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 516–533, 2021.

[176] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[177] R. M. Gray and D. L. Neuhoff. Quantization. *IEEE Trans. Inf. Theory*, 44(6):2325–2383, 1998.

[178] S. Guha and P. Francis. Identity trail: Covert surveillance using DNS. In *Privacy Enhancing Technologies, 7th International Symposium, PET 2007 Ottawa, Canada, June 20-22, 2007, Revised Selected Papers*, volume 4776 of *Lecture Notes in Computer Science*, pages 153–166. Springer, 2007.

[179] H. Habib, J. Colnago, V. Gopalakrishnan, S. Pearman, J. Thomas, A. Acquisti, N. Christin, and L. F. Cranor. Away from prying eyes: Analyzing usage and understanding of private browsing. In *Fourteenth Symposium on Usable Privacy and Security, SOUPS 2018, Baltimore, MD, USA, August 12-14, 2018*, pages 159–175. USENIX Association, 2018.

[180] Y. Harchol, D. Bergemann, N. Feamster, E. Friedman, A. Krishnamurthy, A. Panda, S. Ratnasamy, M. Schapira, and S. Shenker. A public option for the core. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 377–389, 2020.

[181] K. M. Henderson, T. April, and J. Livingood. Authoritative DNS-over-TLS Operational Considerations. Internet-Draft draft-hal-adot-operational-considerations-02, Internet Engineering Task Force, Aug. 2019. Work in Progress.

[182] A. Henzinger, E. Dauterman, H. Corrigan-Gibbs, and N. Zeldovich. Private web search with tiptoe. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 396–416. ACM, 2023.

[183] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. Cryptology ePrint Archive, Paper 2022/949, 2022. https://eprint.iacr.org/2022/949.

[184] D. Herrmann, K.-P. Fuchs, J. Lindemann, and H. Federrath. Encdns: A lightweight privacy-preserving name resolution service. In *European Symposium on Research in Computer Security*, pages 37–55. Springer, 2014.

[185] T. Hiraide and S. Kasahara. Analysis of interaction between miner decision making and user action for incentive mechanism of bitcoin blockchain. *Frontiers Blockchain*, 6, 2023.

[186] N. P. Hoang, I. Lin, S. Ghavamnia, and M. Polychronakis. K-resolver: Towards decentralizing encrypted DNS resolution. *CoRR*, abs/2001.08901, 2020.

[187] P. E. Hoffman and P. McManus. DNS Queries over HTTPS (DoH). RFC 8484, Oct. 2018.

[188] P. E. Hoffman and P. van Dijk. Recursive to Authoritative DNS with Unauthenticated Encryption. Internet-Draft draft-ietf-dprive-opportunistic-adotq-02, Internet Engineering Task Force, Apr. 2021. Work in Progress.

[189] S. Hohenberger, S. Myers, R. Pass, et al. Anonize: A large-scale anonymous survey system. In *2014 IEEE Symposium on Security and Privacy*, pages 375–389. IEEE, 2014.

[190] C.-Y. Hong, S. Mandal, M. Al-Fares, M. Zhu, R. Alimi, C. Bhagat, S. Jain, J. Kaimal, S. Liang, K. Mendelev, et al. B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined wan. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 74–87, 2018.

[191] A. Hounsel, K. Borgolte, P. Schmitt, and N. Feamster. D-DNS: towards re-decentralizing the DNS. *CoRR*, abs/2002.09055, 2020.

[192] A. Hounsel, P. Schmitt, K. Borgolte, and N. Feamster. Encryption without centralization: distributing dns queries across recursive resolvers. In *Proceedings of the Applied Networking Research Workshop*, pages 62–68, 2021.

[193] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. E. Hoffman. Specification for DNS over Transport Layer Security (TLS). RFC 7858, May 2016.

[194] P. Huang, X. Zhang, L. Guo, and M. Li. Incentivizing crowdsensing-based noise monitoring with differentially-private locations. *IEEE Trans. Mob. Comput.*, 20(2):519–532, 2021.

[195] N. Ivkin, Z. Yu, V. Braverman, and X. Jin. Qpipe: Quantiles sketch fully in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 285–291, 2019.

[196] J. Iyengar and M. Thomson. QUIC: A udp-based multiplexed and secure transport. *RFC*, 9000:1–151, 2021.

[197] X. Jin, C. Katsis, F. Sang, J. Sun, E. Bertino, R. R. Kompella, and A. Kundu. Prometheus: Infrastructure security posture analysis with ai-generated attack graphs. *CoRR*, abs/2312.13119, 2023.

[198] X. Jin, C. Katsis, F. Sang, J. Sun, A. Kundu, and R. Kompella. Edge security: Challenges and issues. *CoRR*, abs/2206.07164, 2022.

[199] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, and I. Stoica. Netchain: Scale-free sub-rtt coordination. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 35–49, 2018.

[200] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 121–136, 2017.

[201] X. Jin, S. Manandhar, K. Kafle, Z. Lin, and A. Nadkarni. Understanding iot security from a market-scale perspective. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1615–1629. ACM, 2022.

[202] A. Jonathan, A. Chandra, and J. Weissman. Multi-query optimization in wide-area streaming analytics. In *Proceedings of the ACM symposium on cloud computing*, pages 412–425, 2018.

[203] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[204] D. Kales, O. Omolola, and S. Ramacher. Revisiting User Privacy for Certificate Transparency. In *2019 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 432–447, 2019.

[205] J. Katz and Y. Lindell. *Introduction to modern cryptography*. CRC press, 2020.

[206] M. T. Khan, J. DeBlasio, G. M. Voelker, A. C. Snoeren, C. Kanich, and N. Vallina-Rodriguez. An empirical analysis of the commercial VPN ecosystem. In *Proceedings of the Internet Measurement Conference 2018, IMC 2018, Boston, MA, USA, October 31 - November 02, 2018*, pages 443–456. ACM, 2018.

[207] H. Kim, K. C. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee. Internet traffic classification demystified: myths, caveats, and the best practices. In *Proceedings of the 2008 ACM Conference on Emerging Network Experiment and Technology, CoNEXT 2008, Madrid, Spain, December 9-12, 2008*, page 11. ACM, 2008.

[208] E. Kinnear, P. McManus, T. Pauly, T. Verma, and C. A. Wood. Oblivious DNS over HTTPS. RFC 9230, June 2022.

[209] D. Kogan and H. Corrigan-Gibbs. Private Blocklist Lookups with Checklist. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 875–892. USENIX Association, Aug. 2021.

[210] T. Kolajo, O. Daramola, and A. Adebiyi. Big data stream analysis: a systematic literature review. *Journal of Big Data*, 6(1):1–30, 2019.

[211] M. Kosek, T. V. Doan, S. Huber, and V. Bajpai. Measuring dns over tcp in the era of increasing dns response sizes: A view from the edge. *ACM SIGCOMM Computer Communication Review*, 52(2), 2022.

[212] B. Krebs. A Deep Dive on the Recent Widespread DNS Hijacking Attacks, 2019. `https://krebsonsecurity.com/2019/02/a-deep-dive-on-the-recent-widespread-dns-hijacking-attacks/`.

[213] D. M. Kristol. HTTP cookies: Standards, privacy, and politics. *ACM Trans. Internet Techn.*, 1(2):151–198, 2001.

[214] L. Kugler. How the internet spans the globe. *Communications of the ACM*, 63(1):14–16, 2019.

[215] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja. Twitter heron: Stream processing at scale. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 239–250, 2015.

[216] R. Kumar and F. E. Bustamante. Reclaiming privacy and performance over centralized DNS. *CoRR*, abs/2302.13274, 2023.

[217] J. Kurihara and T. Kubo. Mutualized oblivious DNS (μ ODNS): Hiding a tree in the wild forest. *arXiv preprint arXiv:2104.13785*, 2021.

[218] A. Kwon, D. Lazar, S. Devadas, and B. Ford. Riffle. *Proceedings on Privacy Enhancing Technologies*, 2016(2):115–134, 2016.

[219] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 183–196, 2017.

[220] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. M. Swift. Atp: In-network aggregation for multi-tenant learning. In *NSDI*, pages 741–761, 2021.

[221] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee. Bittorrent is an auction: analyzing and improving bittorrent's incentives. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 243–254, 2008.

[222] S. Li, Z. Yang, N. Hua, P. Liu, X. Zhang, G. Yang, and M. Yang. Collect responsibly but deliver arbitrarily?: A study on cross-user privacy leakage in mobile apps. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1887–1900. ACM, 2022.

[223] W. Li and A. W. Moore. A machine learning approach for efficient traffic classification. In *15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2007), October 24-26, 2007, Istanbul, Turkey*, pages 310–317. IEEE Computer Society, 2007.

[224] Y. Li, R. Miao, C. Kim, and M. Yu. Flowradar: A better netflow for data centers. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 311–324, 2016.

[225] Y. Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *Journal of Cryptology*, 29(2):456–490, 2016.

[226] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114, 2016.

[227] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar. Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[228] K. Loesing, W. Sandmann, C. Wilms, and G. Wirtz. Performance measurements and statistics of tor hidden services. In *Proceedings of the 2008 International Symposium on Applications and the Internet, SAINT 2008, 28 July - 1 August 2008, Turku, Finland*, pages 1–7. IEEE Computer Society, 2008.

[229] Y. LU. Towards plugging privacy leaks in domain name system, cornell university library. *http://arxiv. org/abs/0910.2472*, 2009.

[230] W. Lyu, X. Lin, S. Zheng, L. Pang, H. Ling, S. Jha, and C. Chen. Task-agnostic detector for insertion-based backdoor attacks. *arXiv preprint arXiv:2403.17155*, 2024.

[231] W. Lyu, S. Zheng, H. Ling, and C. Chen. Backdoor attacks against transformers with attention enhancement. In *ICLR 2023 Workshop on Backdoor Attacks and Defenses in Machine Learning*, 2023.

[232] W. Lyu, S. Zheng, T. Ma, and C. Chen. A study of the attention abnormality in trojaned berts. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4727–4741, 2022.

[233] W. Lyu, S. Zheng, T. Ma, H. Ling, and C. Chen. Attention hijacking in trojan transformers. *arXiv preprint arXiv:2208.04946*, 2022.

[234] W. Lyu, S. Zheng, L. Pang, H. Ling, and C. Chen. Attention-enhancing backdoor attacks against bert-based models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10672–10690, 2023.

[235] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6), nov 2013.

[236] Y. Ma, K. Zhong, T. Rabin, and S. Angel. Incremental Offline/Online PIR. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, Aug. 2022. USENIX Association.

[237] J. Magnusson, M. Müller, A. Brunström, and T. Pulls. A second look at DNS QNAME minimization. In *Passive and Active Measurement - 24th International Conference, PAM 2023, Virtual Event, March 21-23, 2023, Proceedings*, volume 13882 of *Lecture Notes in Computer Science*, pages 496–521. Springer, 2023.

[238] Y. Mao and S. B. Venkatakrishnan. Less is more: Understanding network bias in proof-of-work blockchains. *Mathematics*, 11(23):4741, 2023.

[239] Y. Mao and S. B. Venkatakrishnan. Topiary: Fast, scalable publish/subscribe for peer-to-peer (d) apps. *arXiv preprint arXiv:2312.06800*, 2023.

[240] F. J. Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[241] S. J. Menon and D. J. Wu. SPIRAL: Fast, high-rate single-server PIR via FHE composition. In *IEEE S&P*, 2022.

[242] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 15–28, 2017.

[243] M. M. Michael. High performance dynamic lock-free hash tables and list-based sets. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 73–82, 2002.

[244] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani. Deep learning for iot big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials*, 20(4):2923–2960, 2018.

[245] A. Muffett. DNS over Tor. `https://developers.cloudflare.com/1.1.1.1/other-ways-to-use-1.1.1.1/dns-over-tor/`.

[246] A. Muffett. DoHoT: making practical use of DNS over HTTPS over Tor. `https://medium.com/@alecmuffett/dohot-making-practical-use-of-dns-over-https-over-tor-ef58d04ca06a`.

[247] M. H. Mughees, H. Chen, and L. Ren. OnionPIR: Response Efficient Single-Server PIR. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2292–2306, 2021.

[248] M. H. Mughees and L. Ren. Vectorized batch private information retrieval. *Cryptology ePrint Archive*, 2022.

[249] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh. Privacy-preserving matrix factorization. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 801–812, 2013.

[250] D. Nobori and Y. Shinjo. Vpn gate: A volunteer-organized public vpn relay system with blocking resistance for bypassing government censorship firewalls. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI'14)*, pages 229–241, 2014.

[251] A. Panchenko, L. Pimenidis, and J. Renner. Performance analysis of anonymous communication channels provided by tor. In *Proceedings of the The Third International Conference on Availability, Reliability and Security, ARES 2008, March 4-7, 2008, Technical University of Catalonia, Barcelona , Spain*, pages 221–228. IEEE Computer Society, 2008.

[252] P. Papadopoulos, N. Kourtellis, and E. P. Markatos. Cookie synchronization: Everything you always wanted to know but were afraid to ask. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 1432–1442. ACM, 2019.

[253] C. Partridge, T. Mendez, and W. Milliken. Host anycasting service. Technical report, 1993.

[254] N. Patel. Tor networking vulnerabilities and breaches. 2016.

[255] L. Pernal-Stoddart. The Camel's Back: Recursive to Authoritative DNS with Encryption. `https://www.centr.org/news/blog/ietf110-camel-back.html`.

[256] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in bittorrent. In *Proc. of NSDI*, volume 7, page 4, 2007.

[257] I. Poese, S. Uhlig, M. A. Kaafar, B. Donnet, and B. Gueye. Ip geolocation databases: Unreliable? *ACM SIGCOMM Computer Communication Review*, 41(2):53–56, 2011.

[258] R. A. Popa, A. J. Blumberg, H. Balakrishnan, and F. H. Li. Privacy and accountability for location-based aggregate statistics. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 653–666, 2011.

[259] D. R. Ports and J. Nelson. When should the network be the computer? In *Proceedings of the Workshop on Hot Topics in Operating Systems*, pages 209–215, 2019.

[260] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica. Low latency geo-distributed data analytics. *ACM SIGCOMM Computer Communication Review*, 45(4):421–434, 2015.

[261] F. Qi, Y. Chen, M. Li, Y. Yao, Z. Liu, and M. Sun. Onion: A simple and effective defense against textual backdoor attacks. *arXiv preprint arXiv:2011.10369*, 2020.

[262] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman. Aggregation and degradation in jetstream: Streaming analytics in the wide area. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, pages 275–288, 2014.

[263] A. Randall, E. Liu, G. Akiwate, R. Padmanabhan, G. M. Voelker, S. Savage, and A. Schulman. Trufflehunter: Cache snooping rare domains at large public DNS resolvers. In *IMC '20: ACM Internet Measurement Conference, Virtual Event, USA, October 27-29, 2020*, pages 50–64. ACM, 2020.

[264] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), sep 2009.

[265] E. Rescorla. HTTP over TLS. *RFC*, 2818:1–7, 2000.

[266] E. Rescorla. Rfc 8446: The transport layer security (tls) protocol version 1.3, 2018.

[267] S. Rivera, V. K. Gurbani, S. Lagraa, A. K. Iannillo, and R. State. Leveraging ebpf to preserve user privacy for dns, dot, and doh queries. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pages 1–10, 2020.

[268] G. Rosston, S. Savage, and D. Waldman. Household demand for broadband internet service, 2010. `https://siepr.stanford.edu/research/publications/household-demand-broadband-internet-service`.

[269] E. Roth, K. Newatia, Y. Ma, K. Zhong, S. Angel, and A. Haeberlen. Mycelium: Large-scale distributed graph queries with differential privacy. In *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021*, pages 327–343. ACM, 2021.

[270] M. Sabt, M. Achemlal, and A. Bouabdallah. Trusted execution environment: What it is, and what it is not. In *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 1*, pages 57–64. IEEE, 2015.

[271] N. Samardzic, A. Feldmann, A. Krastev, S. Devadas, R. Dreslinski, C. Peikert, and D. Sanchez. F1: A fast and programmable accelerator for fully homomorphic encryption. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 238–252, New York, NY, USA, 2021. ACM.

[272] M. A. Sánchez, J. S. Otto, Z. S. Bischof, D. R. Choffnes, F. E. Bustamante, B. Krishnamurthy, and W. Willinger. Dasu: Pushing experiments to the internet's edge. In *10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pages 487–499, 2013.

[273] I. Sánchez-Rola, M. Dell'Amico, P. Kotzias, D. Balzarotti, L. Bilge, P. Vervier, and I. Santos. Can I opt out yet?: GDPR and the global illusion of cookie control. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*, pages 340–351. ACM, 2019.

[274] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtarik. Scaling distributed machine learning with in-network aggregation. In *18th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 21)*, pages 785–808, 2021.

[275] S. J. Savage and D. Waldman. Broadband internet access, awareness, and use: Analysis of united states household data. *Telecommunications Policy*, 29(8):615–633, 2005.

[276] D. Schmidt, C. Tagliaro, K. Borgolte, and M. Lindorfer. Iotflow: Inferring iot device behavior at scale through static mobile companion app analysis. In *Proceedings of the 2023 ACM*

SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023, pages 681–695. ACM, 2023.

[277] P. Schmitt, A. Edmundson, A. Mankin, and N. Feamster. Oblivious DNS: Practical privacy for DNS queries. *Proceedings on Privacy Enhancing Technologies*, 2019(2):228–244, 2019.

[278] K. Schomp, M. Allman, and M. Rabinovich. DNS resolvers considered harmful. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks, HotNets-XIII, Los Angeles, CA, USA, October 27-28, 2014*, pages 16:1–16:7. ACM, 2014.

[279] K. Schomp, O. Bhardwaj, E. Kurdoglu, M. Muhaimen, and R. K. Sitaraman. Akamai DNS: providing authoritative answers to the world's queries. In *SIGCOMM'20, Virtual Event, USA, August 10-14, 2020*, pages 465–478. ACM, 2020.

[280] K. Schomp, M. Rabinovich, and M. Allman. Towards a model of dns client behavior. In *International Conference on Passive and Active Network Measurement*, pages 263–275. Springer, 2016.

[281] C. Scott, P. Wolfe, and M. Erwin. *Virtual private networks*. " O'Reilly Media, Inc.", 1999.

[282] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015*, pages 213–226. ACM, 2015.

[283] N. Shetty, G. Schwartz, and J. Walrand. Internet qos and regulations. *IEEE/ACM Transactions on Networking*, 18(6):1725–1737, 2010.

[284] E. Shi, W. Aqeel, B. Chandrasekaran, and B. Maggs. Puncturable pseudorandom sets and private information retrieval with near-optimal online bandwidth and time. In *Annual International Cryptology Conference*, pages 641–669. Springer, 2021.

[285] S. Singanamalla, S. Chunhapanya, J. Hoyland, M. Vavrusa, T. Verma, P. Wu, M. Fayed, K. Heimerl, N. Sullivan, and C. A. Wood. Oblivious DNS over HTTPS (ODoH): A Practical Privacy Enhancement to DNS. *Proc. Priv. Enhancing Technol.*, 2021(4):575–592, 2021.

[286] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking. Packet transactions: High-level programming for line-rate switches. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 15–28, 2016.

[287] R. N. Staff. Ripe atlas: A global internet measurement network. *Internet Protocol Journal*, 18(3):2–26, 2015.

[288] R. N. Staff. Ripe atlas: A global internet measurement network. *Internet Protocol Journal*, 18(3), 2015.

[289] H. Sun, Y. Xiao, J. Wang, J. Wang, Q. Qi, J. Liao, and X. Liu. Common knowledge based and one-shot learning enabled multi-task traffic classification. *IEEE Access*, 7:39485–39495, 2019.

[290] J. Sun, P. Tang, and Y. Zeng. Games of miners. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*, pages 1323–1331. International Foundation for Autonomous Agents and Multiagent Systems, 2020.

[291] Q. Sun, D. R. Simon, Y. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical identification of encrypted web browsing traffic. In *2002 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 12-15, 2002*, pages 19–30. IEEE Computer Society, 2002.

[292] H. Taneja and A. X. Wu. Does the great firewall really isolate the chinese? integrating access blockage with cultural factors to explain web user behavior. *The Information Society*, 30(5):297–309, 2014.

[293] A. D. P. Team. Learning Privacy at Scale, 2021. `https://machinelearning.apple.com/research/learning-with-privacy-at-scale`.

[294] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156, 2014.

[295] M. Trusov, L. Ma, and Z. Jamal. Crumbs of the cookie: User profiling in customer-base analysis and behavioral targeting. *Mark. Sci.*, 35(3):405–426, 2016.

[296] P. Vallina, V. Le Pochat, Á. Feal, M. Paraschiv, J. Gamba, T. Burke, O. Hohlfeld, J. Tapiador, and N. Vallina-Rodriguez. Mis-shapes, mistakes, misfits: An analysis of domain classification services. In *Proceedings of the ACM Internet Measurement Conference*, pages 598–618, 2020.

[297] H. R. Varian. The demand for bandwidth: evidence from the index project, 2014.

[298] D. Vekshin, K. Hynek, and T. Cejka. Doh insight: detecting DNS over HTTPS by machine learning. In *ARES 2020: The 15th International Conference on Availability, Reliability and Security, Virtual Event, Ireland, August 25-28, 2020*, pages 87:1–87:8. ACM, 2020.

[299] R. Wails, A. Johnson, D. Starin, A. Yerukhimovich, and S. D. Gordon. Stormy: Statistics in tor by measuring securely. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 615–632. ACM, 2019.

[300] H. Wang, R. Soulé, H. T. Dang, K. S. Lee, V. Shrivastav, N. Foster, and H. Weatherspoon. P4fpga: A rapid prototyping framework for p4. In *Proceedings of the Symposium on SDN Research*, pages 122–135, 2017.

[301] L. Wang, H. Kim, P. Mittal, and J. Rexford. Programmable in-network obfuscation of dns traffic. In *NDSS: DNS Privacy Workshop*. sn, 2021.

[302] L. Wang, H. Wang, R. He, R. Tao, G. Meng, X. Luo, and X. Liu. Malradar: Demystifying android malware in the new era. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(2):1–27, 2022.

[303] M. A. Warrior, Y. Xiao, M. Varvello, and A. Kuzmanovic. De-kodi: Understanding the kodi ecosystem. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, pages 1171–1181. ACM / IW3C2, 2020.

[304] M. Wątorek, S. Drożdż, J. Kwapień, L. Minati, P. Oświęcimka, and M. Stanuszek. Multiscale characteristics of the emerging global cryptocurrency market. *Physics Reports*, 901:1–82, 2021.

[305] M. A. Wiering and M. Van Otterlo. Reinforcement learning. *Adaptation, learning, and optimization*, 12(3):729, 2012.

[306] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Dissent in numbers: Making strong anonymity scale. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 179–182, 2012.

[307] D. I. Wolinsky, K. Lee, P. O. Boykin, and R. J. O. Figueiredo. On the design of autonomic, decentralized vpns. In *The 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom 2010, Chicago, IL, USA, 9-12 October 2010*, pages 1–10. ICST / IEEE, 2010.

[308] N. Xia, H. H. Song, Y. Liao, M. Iliofotou, A. Nucci, Z. Zhang, and A. Kuzmanovic. Mosaic: quantifying privacy leakage in mobile networks. In *ACM SIGCOMM 2013 Conference, SIGCOMM 2013, Hong Kong, August 12-16, 2013*, pages 279–290. ACM, 2013.

[309] Y. Xiao, S. Markovich, and A. Kuzmanovic. Blockchain mining: Optimal resource allocation. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies, AFT 2022, Cambridge, MA, USA, September 19-21, 2022*, pages 294–307. ACM, 2022.

[310] Y. Xiao, H. Sun, Z. Zhuang, J. Wang, and Q. Qi. Common knowledge based transfer learning for traffic classification. In *43rd IEEE Conference on Local Computer Networks, LCN 2018, Chicago, IL, USA, October 1-4, 2018*, pages 311–314. IEEE, 2018.

[311] Y. Xiao and M. Varvello. FIAT: frictionless authentication of iot traffic. In *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2022, Roma, Italy, December 6-9, 2022*, pages 156–170. ACM, 2022.

[312] Y. Xiao, M. Varvello, and A. Kuzmanovic. Monetizing spare bandwidth: The case of distributed vpns. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(2):33:1–33:27, 2022.

[313] Y. Xiao, M. Varvello, M. A. Warrior, and A. Kuzmanovic. Decoding the kodi ecosystem. *ACM Trans. Web*, 17(1):2:1–2:36, 2023.

[314] Y. Xiao, C. Weng, R. Yu, P. Liu, M. Varvello, and A. Kuzmanovic. Demo: PDNS: A fully privacy-preserving DNS. In *Proceedings of the ACM SIGCOMM 2023 Conference, ACM SIGCOMM 2023, New York, NY, USA, 10-14 September 2023*, pages 1182–1184. ACM, 2023.

[315] Y. Xiao, Y. Zhao, S. Lin, and A. Kuzmanovic. Snatch: Online streaming analytics at the network edge. In *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024*. ACM, 2024.

[316] Z. Xiong and N. Zilberman. Do switches dream of machine learning? toward in-network classification. In *Proceedings of the 18th ACM workshop on hot topics in networks*, pages 25–33, 2019.

[317] B. Xue, Y. Mao, S. B. Venkatakrishnan, and S. Kannan. Goldfish: Peer selection using matrix completion in unstructured p2p network. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE, 2023.

[318] D. Yang, B. Qu, and P. Cudré-Mauroux. Privacy-preserving social media data publishing for personalized ranking-based recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 31(3):507–520, 2018.

[319] Z. Yang, S. Zhong, and R. N. Wright. Anonymity-preserving data collection. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pages 334–343. ACM, 2005.

[320] Z. Yu, Y. Zhang, V. Braverman, M. Chowdhury, and X. Jin. Netlock: Fast, centralized lock management using programmable switches. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 126–138, 2020.

[321] Y. Yuan, O. Alama, J. Fei, J. Nelson, D. R. K. Ports, A. Sapio, M. Canini, and N. S. Kim. Unlocking the power of inline floating-point operations on programmable switches. In *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*, pages 683–700. USENIX Association, 2022.

[322] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the twenty-fourth ACM symposium on operating systems principles*, pages 423–438, 2013.

[323] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee. Awstream: Adaptive wide-area streaming analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 236–252, 2018.

[324] F. Zhao, Y. Hori, and K. Sakurai. Two-servers PIR based DNS query scheme with privacy-preserving. In *The 2007 International Conference on Intelligent Pervasive Computing (IPC 2007)*, pages 299–302. IEEE, 2007.

[325] M. Zhou, A. Park, E. Shi, and W. Zheng. Piano: Extremely simple, single-server pir with sublinear server computation. Cryptology ePrint Archive, Paper 2023/452, 2023. `https://eprint.iacr.org/2023/452`.

[326] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pages 95–109. IEEE Computer Society, 2012.