An Anomaly Detection Framework for DNS-over-HTTPS (DoH) Tunnel Using Time-series Analysis

by

Mohammadreza MontazeriShatoori

BSc in Software Engineering,
Sharif University of Technology (SUT), 2018

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor(s):	Arash Habibi Lashkari, Ph.D, Computer Science
Examining Board:	Rongxing Lu, Ph.D, Computer Science, Chair
	Suprio Ray, Ph.D, Computer Science
	Mohsen Mohammadi, Ph.D, Faculty of Engineering, UNB

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

December, 2020

Abstract

Domain Name System (DNS) as a network protocol is vulnerable to several security loopholes. To cover up some of the vulnerabilities in DNS, a new protocol, named DNS over HTTPS (DoH), is created to improve privacy, and protect from various persistent attacks. The DoH protocol encrypts the DNS requests for the DoH client and sends it through a tunnel to prevent eavesdropping and man-in-the-middle attacks. This research work comprehensively studies these security vulnerabilities, proposes a taxonomy of potential DNS attacks, analyzes the security aspects of DoH protocol, and classifies DNS attacks that are applicable on DoH. To achieve these objectives, we simulated DoH tunnels. The simulated environment covers different DoH deployment scenarios includes DoH within an application, DoH proxy on the name server in the local network, and DoH proxy on a local system as suggested in RFC8484. In this research, we captured malicious and benign DoH traffic and analyzed it as a two-layered approach to classify benign and malicious traffic at first layer and characterize DoH traffic at second layer. It is observed that for statistical features, Random Forest (RF) and Decision Tree (DT) give the best classification and characterization results among prominent machine learning and deep learning classifiers at first and second layer, respectively. Moreover, for time-series features, long short-term memory (LSTM) turns out to be the best classifier for DoH traffic classification and characterization at first and second layers, respectively. The experimental results indicate that while DoH can be abused to create covert com-

layer1: 区分恶意流量
layer2: 区分恶意流量中的DOH部分。

随机森林+决策树在统计特征中的分类效果最好。

LSTM 在时序特征中分类效果最好。

munication channels, the proposed solution is sufficient to detect these channels in a timely manner.

Acknowledgements

I would like to first sincerely thank my supervisor Prof. Arash Habibi Lashkari for all the support and help he offered me through my program. I also wish to express my gratitude to Dr. Gurdip Kaur who was a great teammate to work with. I also would like to acknowledge the support of Canadian Institute for Cybersecurity (CIC). This work was made possible by the financial support of University of New Brunswick (UNB) and the Canadian Internet Registration Authority (CIRA).

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Domain Name System (DNS) is a naming system mostly known for its importance to provide a mapping between human-readable domain names and computer understandable Internet Protocol (IP) addresses. This protocol can also map domain names to data types other than IP, such as texts, keys, and other domain names. Being one of the earliest network protocols still is used, DNS has several critical security and privacy issues. For example, DNS mostly uses plain-text UDP packets that can be easily accessed by actors that have access to the underlying network. This vulnerability makes DNS protocol highly susceptible to various active and passive attacks, such as man-in-the-middle attacks (MitM) and eavesdropping.

DNS 本质上是 明文
udp包

Moreover, other vulnerabilities in DNS protocol and its implementations have helped attackers to mount attacks that would not target DNS servers and communications, but misuse the DNS protocol to target other servers and networks. An example of such attacks is ”DNS reflection attack” that uses IP spoofing in DNS requests to direct the DNS response packets to the target network, and disables the network by flooding it with these packets. Several updates to the DNS protocol and infrastructure has been made over the years to make it more secure, including DNSSEC, which is a collection of security extensions specified by Internet Engineering Task

Force (IETF) [47]. Other efforts have been made to make DNS more secure and private, including the introduction of new encrypted DNS protocols such as DNSCrypt and DNS-over-TLS[25].

In 2018, a new protocol named DoH was released, which not only enhances performance but also improves user privacy and security by preventing eavesdropping and manipulation of DNS data through MitM attacks [22]. DoH wraps DNS records (both requests and responses) in an HTTPS connection, providing encryption and authentication of the server, and changing the connection-less aspect of DNS. DoH primarily serves two purposes–preventing on-path devices from interfering with DNS operations and allowing web applications to access DNS information via existing browser APIs. It can be deployed in three ways, by installing a proxy on a local system, implementing it within an application, or installing a proxy on the local name server.

DOH 两大作用：
1）避免DNS请求和返回路径上的机器对DNS操作进行修改；
2）允许web应用通过API接口访问DNS返回信息。

DOH使用方式：
1）本地部署doh proxy
2）应用中使用DOH
3）本地域名服务器使用DOH

Even though DoH provides additional security due to the encryption of DNS records, there has not been a thorough investigation of its security vulnerabilities and limitations. Most importantly, while there is an abundance of research on DNS security and the possible attacks on DNS protocol and infrastructure, the corresponding attacks on DoH are not studied systematically. In this thesis, first, all the existing DNS attacks are reviewed, and then the related attacks on DoH protocol are studied. To provide a systematic and thorough investigation of DoH security, I investigated all three different deployment strategies in this work.

本文关注点：回顾所有DNS后，研究DOH相关的攻击。
同时3类不同的DOH使用方式重点关注。

As one of the most critical security concerns of DoH protocol, DoH covert channels are used by attackers to hide malicious DNS queries inside it by tunneling data through DNS packets [42]. Covert channels are discussed and simulated using a series of experiments in this thesis. These covert channels are an upgraded version of the traditional DNS tunnels that are already abused by malware as a covert method of communication. DNS tunneling encapsulates the DNS communication between a

本文实验搭建的convert tunnel是已被malware 控制用于隐秘传输的情况，非正常DNS tunnel

encapsulates 压缩

eavesdropper 窃听者

DNS client and DNS server to make it difficult to interpret by an eavesdropper. The DNS data is coded within records of an otherwise standard DNS request and the server may or may not return with some data encoded in the corresponding DNS response.

不返回对应的请求的情况，即可被用于传输恶意C & C 或 信息泄露。

使用DNS作为数据传输方法的动机：
1）大部分防火墙不去验证DNS请求的频率和内容；
2）经过递归解析后跳转到被控域名服务器的情况难以被检测并通过黑名单防御；

There can be different motives for using DNS as a means of data transmission. Most importantly, many firewalls do not examine the contents and frequency of DNS packets, which makes it easier for this data transmission to go unnoticed. 1 Secondly, since the packet will reach the modified name-servers over several hops through DNS recursion, the communication is harder to detect and block using standard methods such as IP blacklisting. 2 Many malware has already abused this method to exfiltrate data or to create a communication channel with their command-and-control servers. This vulnerability helps attackers to take some degrees of control over the infected devices and potentially use them to orchestrate attacks on other targets such as launching DDoS attacks on online services [38].

DNS tunnel检测方案

DNS tunnel feature：
1）DNS 解析频率
2）子域名长度
3）TXT record使用方式

DNS tunnels can be detected with a variety of methods including domain name analysis and statistical analysis of DNS packets. Different approaches have been proposed to detect anomalous domain names, such as using statistical models [11, 18, 17, 38]. Based on the previous research, frequency of DNS resolutions, length of the sub-domains, and the usage of TXT records are also common variables that can help detect DNS tunnels [11, 42]. Also, blocking DNS tunnels may be achieved through domain blacklisting, IP blacklisting, and dropping DNS packets that are thought to be malicious [15].

域名、ip黑名单、丢弃DNS恶意包的方式阻断DNS隧道。

Many security researchers have been criticized DoH for making DNS tunnels harder to detect and mitigate. First, Since the DoH wraps the DNS traffic in HTTPS, the DNS traffic is imperceptible to the network infrastructure between the client (malware) and the DoH server. This feature effectively makes detection methods that rely on examining the DNS packets obsolete for the firewalls. Second, since

1、Doh 将 DNS traffic 放入HTTPS，DNS traffic 在网络基础设置中难以被识别，导致防火墙检测DNS包的方案失效。

doh making DNS tunnels harder to detect

2、使用的http/2是DOH 标准推荐的版本，恶意软件能使用http/2 连接来实现 “ 一次连接传输多个doh请求和响应，隐藏
frequency of DNS resolution." 进一步减少探测DNS tunnels 的方法。

HTTP/2 is the minimum version of HTTP that DoH standard recommends for using with DoH, Malware can utilize the HTTP/2 connection to send several DoH request, without creating a separate connection (or packet) for each request. The same also applies to the responses that DoH server is sending to the malware. Through this method, malware can hide the frequency of their DNS resolutions, further reducing the number of methods that can detect DNS tunnels [1].

By simulating the network traffic made by DoH covert channels in this work, I provide a way of studying their properties and relevant mitigation strategies that can be used to limit their potential adverse effects. The network traffic from these simulations is captured in the form of a dataset that would help us (and other researchers) to create classifiers capable of detecting DoH covert channels. In addition to using various DNS tunneling tools and making them compatible with DoH protocol, a proof-of-concept DoH tunneling tool, in the form of a bot, is also created and used in creating the covert channels included in the dataset. All datasets and developed packages are publicly available as free datasets and open-source projects.

和前面的两侧分类
顺序刚好相反，笔误？

Detection of DoH tunnels involves a two-layer classifier that first labels traffic as DoH or non-DoH (any other HTTPS traffic), and then classifies the DoH part of the dataset into two classes of benign or malicious DoH traffic. To find the best set of features and algorithms for this classification attempts, a variety of classifiers and the features those classifiers work with, are studied based on the existing literature on encrypted network traffic classification. It is shown that the proposed classification technique using time-series features can perform as well as the current methods while needing only a small subset of traffic compared to those methods.

1.1 Summary of Contributions

In this thesis, the following contributions are made:

构建已知DNS攻击并

- Creating a taxonomy of known DNS attacks ($Cont_1$).

- Systematically investigating the related attacks on DoH protocol and finding the weak spots on the design of this protocol ($Cont_2$).

- Presenting a novel two-layered approach to classify DoH traffic from non-DoH traffic at layer 1 and characterize DoH traffic at layer 2 ($Cont_3$).

- Proposing a new feature-based DoH anomaly detection framework using time-series representation of traffic flows by introducing the concept of packet clumps and demonstrating the effectiveness of this feature set in encrypted traffic characterization ($Cont_4$).

- Generating a labeled dataset by capturing Benign-DoH, Malicious-DoH and non-DoH encrypted traffic in the network premises and evaluate the proposed detection framework ($Cont_5$). 良性DOH，恶意性DOH，非DOH 加密留咯数据集

1.2 Thesis Organization

The following chapters of this thesis are organized as follows:

- In chapter 2, first the necessary background for the thesis is established by detailing DNS and DoH protocols. Then, a taxonomy of the known DNS attacks is created by reviewing the existing literature on DNS vulnerabilities. This taxonomy is then used to find related attacks on DoH. Finally, the current literature on DoH attacks is reviewed, plus all the relevant studies that would help us in designing and evaluating my mitigation technique.

- Chapter 3, describes the framework designed to counter DoH tunnels.

- Chapter 4 details the implementation of the proposed framework, including the technical steps taken to create the necessary dataset and to implement the classifiers. Furthermore, the implementation details of DoHBot are presented in this chapter.

- Chapter 5 focuses on the experiments done to evaluate the detection framework. Details of the experiments that compare the proposed method to the existing techniques are given, and their results are analyzed to provide a clear view of the pros and cons of the proposed framework. Using the abilities included in DoHBot, rule of padding in preventing detection of DoH tunnels are also evaluated.
- Chapter 6 summarizes the contributions of this work and provides possible future works.

Chapter 2

Background and Related Works

2.1 Domain Name System

Domain Name System (DNS) is one of the most important protocols of the Internet, used for providing a mapping between human-readable hostnames and computer understandable Internet Protocol (IP) addresses. DNS protocol uses a decentralized hierarchical approach that helps this protocol scale with the growth of the Internet. To provide this decentralization while maintaining the authenticity of the results, DNS defines hierarchical divisions known as DNS zones that are distributed between name servers. Each zone can be specified by a domain name (such as ”example.com.”) that also shows what subset of DNS queries can be answered by the name servers responsible for that zone. Furthermore, name servers responsible for a zone can delegate the authority of each of their sub-domains to other servers, thus performing “cuts” that create new zones. For example, the name servers for “example.com.” can delegate the authority of “test.example.com.” to some other name servers.

The root of this tree-like structure is called the DNS root zone, which has the label ‘.’ (dot). The servers responsible for answering queries in this zone are called root name servers. There are 13 root name servers as of July 2020, controlled by various organizations, all under the oversight of Internet Assigned Numbers Authority (IANA).

When a DNS client generates a DNS query asking for an IP address, the local DNS server responds after looking into its cache. Suppose it does not find the answer within the cache memory. In this case, it forwards the DNS query to recursive DNS resolver. Then it tracks down the DNS record with repetitive DNS queries to root name servers, Top Level Domain (TLD) name servers and authoritative name servers until it gets the target authoritative answer [36]. Figure 2.1 shows the working of the domain name system with sequence number marked on the direction of flow.

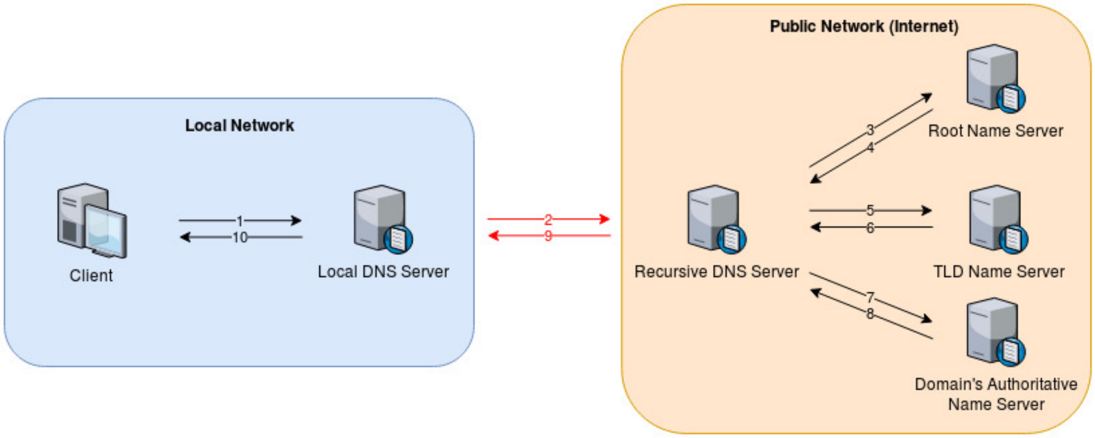


Figure 2.1: Domain Name System

2.2 DNS Vulnerabilities

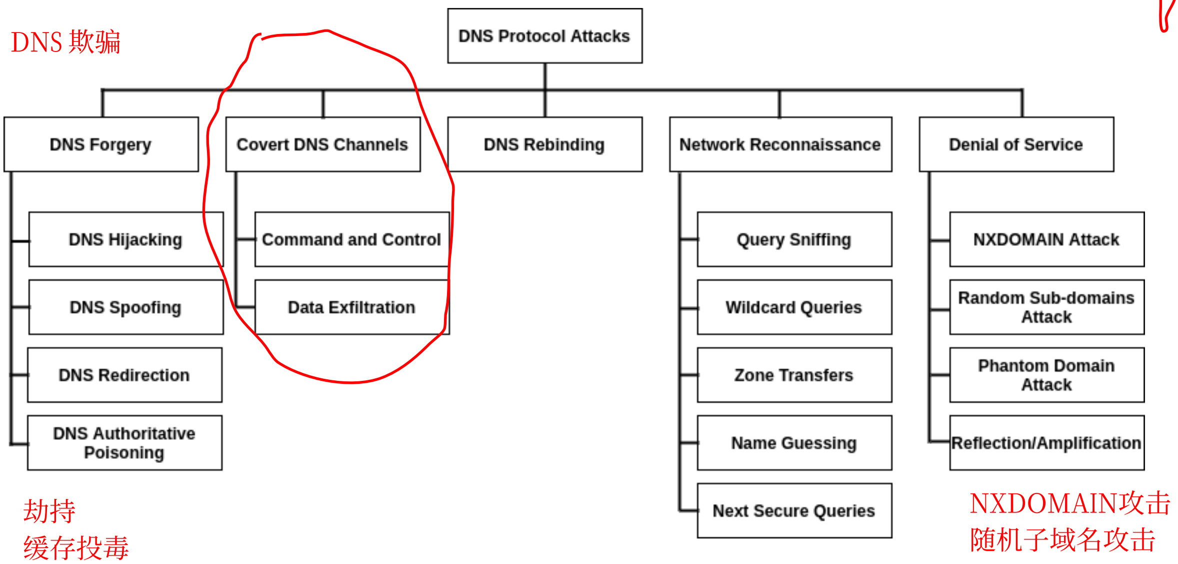
To have a comprehensive view of DNS security concerns, I examine both attacks that target DNS infrastructure and individual users using DNS and attacks where DNS protocol is itself not targeted by malicious actors but is abused in the process of the attack. It is worth noting that even though the academic literature on DNS

3类攻击：1）攻击DNS的基础设置(解析服务器劫持、缓存投毒等）、2）DNS用户被解析到恶意网站、3）利用DNS进行恶意行为。

security regards each of these attacks separately, in real life cyberattacks may use a combination of DNS vulnerabilities to create attacks based on their motives, abilities and the situation in hand.

There are multiple ways of categorizing cyber-attacks related to the DNS protocol. In this section, I classify these attacks by the type of malicious activity they incorporate, and provide a taxonomy of DNS attacks as shown in Figure 2.2 ($Cont_1$ covered).

DNS 欺骗



劫持
缓存投毒
重定向
权威服务器投毒

NXDOMAIN攻击
随机子域名攻击
幻影域攻击
反射域/放大域攻击

Figure 2.2: Taxonomy of DNS attacks

2.2.1 DNS Forgery

DNS Forgery is any activity that is done to introduce unauthorized changes to the DNS responses. Some sources call these activities DNS Poisoning, but I found DNS Forgery to be a better umbrella term since usually DNS Poisoning refers just to DNS Cache Poisoning. These activities include DNS Hijacking, DNS Spoofing (also known as DNS Cache Poisoning), DNS Redirection, and DNS Authoritative Poisoning.

2.2.1.1 DNS Hijacking

DNS服务器返回被修改后的DNS信息，类似DNS重定向。

This type of activity involves a DNS server responding with fake DNS information. Even though the same term has been used for what I call DNS Redirection in this work, I use DNS Hijacking strictly to refer the act of sending fake DNS responses by DNS servers. In other words, I do not consider attacks that change the DNS resolver used by the user as DNS Hijacking (even though they may also incorporate DNS Hijacking). Non-malicious uses of DNS Hijacking include redirecting users to captive portals (used widely in open wireless networks), censorship by governments, blocking malicious websites, and displaying customized error pages (sometimes including ads) instead of responding with NXDOMAIN. It can also be incorporated by rogue DNS servers to redirect users to malicious websites such as the ones used for phishing.

2.2.1.2 DNS Spoofing (DNS Cache Poisoning)

也可以故意进行缓存投毒

This attack works by introducing forged DNS information into the cache of DNS resolvers, causing the resolver to mistakenly send potentially malicious responses to the queries made by users [54]. To produce such an effect, the malicious actor should make the DNS resolver believe that it had received a valid response to the requested query. To do this, they should send forged DNS responses to the DNS requests made by the resolver, with the same query ID and port number as the DNS request packet. The malicious actor may have the necessary access to the network for performing packet interceptions, allowing them to look at the DNS requests made by resolver (and their port number and query ID). Otherwise, they would have to guess those values, so that their responses would not get dropped by the resolver. In addition to the main attack that involves poisoning a single DNS entry (domain), a variation of this attack mostly referred to as *Kaminskey Attack* [20] can poison a whole zone (a domain and all of its subdomains).

通过提前返回同样的请求ID和投毒的内容让递归服务器缓存错误的信息。(正确信息来了发现已有回答了就丢弃)

2.2.1.3 DNS Redirection

In this type of attack, that is also known as “Resolver Redirection Attack”, the goal of the attacker is to change the default DNS resolver of the user to a name server controlled by them. This attack is usually performed through malware or by changing DNS resolver settings on vulnerable network infrastructure, such as routers with default passwords that are accessible through the network. The resolver configuration may be performed manually or through DHCP/PPP [15]. This change will let the attacker control all of DNS responses so that they could collect information and/or possibly redirecting users to malicious web pages. What makes this attack, especially dangerous is that the user does not get any indication of the attack. By hijacking the DNS responses, the attacker can use DNS redirection to enable more sophisticated attacks, such as phishing, with minimum risk of detection.

2.2.1.4 DNS Authoritative Poisoning

This attack focuses on changing DNS records on the DNS authoritative servers. While this change would be more challenging than changing the resolver settings of users, it has the advantage of affecting all of the users at once. To perform this type of attack, the attacker would need to focus on the vulnerabilities in the implementation of DNS name servers. Such vulnerabilities may give enough access to the attacker to either change the records or redirect part of the DNS zone to another name server controlled by the attacker [15].

2.2.2 Covert DNS Channels

DNS tunneling is a method of using DNS protocol as a mean of encapsulating data transmission between a client and a server. The data is coded within parts of an otherwise standard DNS request and the server (which is designed to understand these requests) may or may not return with some data encoded in the corresponding

只是信息传输，就不需要DNS响应。

DNS response. There can be different motives for using DNS as a means of data transmission. Most importantly, many ① firewalls don't examine the contents and frequency of DNS packets, which makes it easier for malware to go unnoticed [51]. ② Secondly, since the packet will reach the modified nameservers over several hops through DNS recursion, the communication is harder to detect and block, using standard methods such as IP blacklisting. Many malware samples have already used this method to exfiltrate data [45, 58, 37], or to create a communication channel with their Command-and-Control servers [14]. This option helps attackers to take some degrees of control over the infected devices and potentially using them to orchestrate attacks on other targets, for example, launching DDoS attacks on online services.

1）通过DNS tunnel向外输出的信息，多采用base32加密后放入DNS域名的子域名，分多次发出(每次小于253字节-域名最大长度)，多个请求接收后即为最终信息。

2）向内传输信息or命令：通过base64或类似加密方式，放入DNS response的 TXT记录中.

The outgoing data is encoded with a particular encoding such as base32 that only produce outputs valid for a subdomain. Then a DNS packet is created using the encoded data as the subdomain of an address that is going to be resolved. By sending several DNS requests, each containing less than 253 bytes of the data (the maximum length of a full domain name), the malware can send varying amount of data to the attacker. The incoming data is usually encoded with base64 or similar encodings and put into a TXT record of a DNS response. This data may be used to control the infected device.

There are multiple ways to detect and shut down DNS tunnels which usually try to investigate the domain names used in DNS requests. Different approaches have been used to detect anomalous domain names, such as using statistical models [11, 17, 18, 38]. Frequency of DNS resolutions, length of the subdomains, and the usage of TXT records are also variables that can help detect DNS tunnels [9, 44]. Blocking DNS tunnels may be achieved through domain blacklisting, IP blacklisting, and dropping DNS packets that are thought to be malicious.

2.2.3 DNS Rebinding

DNS rebinding attacks bypass the local network restrictions by mapping a domain, which is controlled by the attacker to a host in the local network. These attacks are used to exfiltrate sensitive information, breach privacy networks, perform privileged operations, and trigger remote code execution to exploit a vulnerable service on network [35]. An attacker registers a domain and delegates it to a DNS server under his control. The server is configured in such a way that it does not save the response in its cache. When a victim browses the malicious domain, he is redirected to the server hosting malicious content. The exploitation of vulnerable services can be simplified by using *Singularity of Origin*, a tool publicly available in GitHub [39]. However, this attack can be easily prevented by following techniques:

https://github.com/nccgroup/singularity

- Filtering private and loopback IP addresses
- Using a firewall in the gateway or on local computer
- Configuring web browsers to resist DNS rebinding
- Configuring web servers to drop HTTP requests from unrecognized hosts

2.2.4 Network Reconnaissance

To attack the desired target, attackers need to access the DNS repository to obtain the victim's namespace information. Following reconnaissance methods can be used by attackers to do so[15]:

- Query Sniffing: An attacker can log DNS queries and responses to identify the potential targets. 查询嗅探
- Wildcard Queries: Attackers can address their queries with ANY wildcard so that DNS server responds with all the resource records associated with a queried domain. 通配符查询

- Zone Transfers: Attacker impersonates as a slave server and requests zone transfer from the master server to identify targets for direct attacks. Zone transfers include information such as name servers (NS), hostnames (A), aliases (CNAME), mail exchangers (MX), the Start Of Authority (SOA) records, pointer records (PTR), and so on[43].

2.2.5 Denial of Service

There are several types of flood attacks that can cripple DNS servers. The main idea is to flood a DNS server (either recursive or authoritative, or both) with DNS requests that are crafted to use crucial resources such as CPU, memory, and cache space. If effective, this attack can slow down the target, thus preventing the resolution of legitimate DNS requests.

DNS servers can mitigate this kind of attacks by several methods, including:

- Preventing cache pollution with better cache management, such as not keeping NXDOMAIN responses.

洪泛攻击解决方法：
1）优化缓存存储逻辑，比如不再存储NXdomain response
2）将查询NXDOMAIN的恶意请求client加入黑名单

- Detecting malicious clients with the number of NXDOMAIN requests and blacklisting them.

- Blacklisting hosts that are slow to respond.

DoH usage can affect these attack strategies in many ways. Floods that target DoH servers may be more effective in comparison with their DNS counterpart. DoH servers keep an open HTTP/2 connection with their clients that also have the overhead of HTTPS handshaking. Attackers can create particular malware that maximizes this overhead on DoH servers, thus making DoH flood attacks more dangerous that DNS flood attacks.

DOH 更容易受到洪泛攻击的影响，因为HTTP/2的连接涉及多次握手，可以通过请求增大握手次数，降低DOH server的响应速度等。

2.2.5.1 NXDOMAIN attack

In the NXDOMAIN variant, the attacker uses domain names that don't exist. These prompts the recursive server to do DNS recursion. By flooding the server with these requests, resources such as CPU, and memory get exhausted, and the server slows down. Depending on the implementation of the DNS server, it can also fill up the DNS cache with NXDOMAIN responses and flush out other cache entries that are legitimate. This process is called cache pollution, and it forces the DNS server to do more recursions (since the number of cache hits goes down drastically in this state).

2.2.5.2 Random subdomains attack

In the Random Subdomain variant, the attacker uses addresses that are random subdomains of a legitimate domain. Like the NXDOMAIN attack, the recursive server still needs to do recursion and experiences the load of the flood of DNS requests. But the flood also incapacitates the authoritative servers of the domain used, by the same mechanism.

Where DoH server itself is not the target of a flood attack, there are advantages for the attacker in the Random Subdomain variant of the attack. Since the traffic is funneled through the DoH server, the DNS request would be stripped of data that can identify sources of attack. So, the target server cannot detect and blacklist malicious clients. The attack can be mitigated by dropping all packets from the DoH server. But if the usage of DoH instead of DNS is widespread, which means legitimate DNS traffic is also dropped and will deny normal users from using the DNS server.

在使用常规DNS server的情况下，可以将所有DOH server的请求做返回即可，在使用DOH的情况下，

2.2.5.3 Phantom domain attack

幻影域攻击

In the Phantom Domain variant, the attacker sets up special name servers that are designed to be slow, or unresponsive. The flood will target the recursive server with

向目标递归服务器发送必须在设定好的很慢or不响应的递归服务器上解析的DNS请求，从而实现recursive server 资源浪费。

requests that should be resolved using the particular name server (for example, using random subdomains of the domain of that server). The specific name server ensures that the recursive server wastes a substantial amount of resources for each request. By flooding the name server with these time-wasting requests, the attacker wishes to exhaust the available resources on the victim's name server, eventually making it impossible for legitimate queries to go through [15].

2.2.5.4 Reflection/Amplification attack

将控制者所控制的各种DNS解析器获得的多个解析结果一起发送给victim 的一个DNS 请求。

DNS amplification/reflection attack abuses the connection-less nature of DNS to spoof the sender IP of a DNS query, sending the results of all queries he/she makes from a wide range of clients to the victim's server [48]. The attacker also may amplify these responses by creating DNS queries with much larger responses that the query packet size. This flood of packets may cripple the victim's local network, rendering the victim's server unresponsive. Several mitigation techniques have been proposed for this problem [13, 26, 48], but since this issue stems from the original design of DNS protocol, we can not completely rule out these attacks. A recent variant of this attack, called NXNSAttack, abuses the limitations in DNS server implementations to amplify and reflect seemingly legitimate DNS packets to the target servers [50].

2.3 DNS Available Security Solutions

As I discussed in the previous sections, the DNS protocol comes with a lot of risks in security and privacy, mostly due to the unencrypted nature of the contents of DNS packets. These risks mostly arise from the fact that DNS is one of the oldest protocols of the Internet and has been designed without regards to the security standards and solution that are well-accepted nowadays, such as digital signatures or cryptography. DNS is widely used on the Internet for a variety of applications, mainly domain resolution. This widespread usage of DNS makes changes to the DNS protocol a challenge since changes should be backwards-compatible to make sure they don't break any of the existing services utilising DNS. There have been several attempts to make DNS more secure with varying levels of success. DNSSEC (Domain Name System Security Extensions) and DNS-over-encryption are currently seen as the leading solutions to DNS security issues.

2.3.1 DNSSEC

To establish authenticity of DNS responses, IETF introduced a set of extensions to DNS called DNSSEC. These new specifications allow for backwards-compatible additions to DNS, providing cryptographic authentication of data received by clients. RFC3833 outlined the explicit design requirement for DNSSEC as data integrity and data origin authentication [7]. To deliver these requirements, DNSSEC uses digital signatures in the form of additional resource records (RR) included in DNS responses. These resource records provide signatures for the response and the cryptographic keys necessary for validating the signature. The digital signature used by DNSSEC utilizes public-key cryptography and authentication is done through a chain of trust, starting at DNS root zone [4].

数字签名基于信任链实现，从DNS 根域开始。

Even though DNSSEC will provide data integrity for clients that support it, it still won't provide any additional security for confidentiality of the transmitted data. DNSSEC authenticates the DNS messages but will not encrypt them. So we have to implement other solutions in DNS infrastructure to provide confidentiality of the data.

DNSSEC 仅做验证，不做加密，最终的DNS响应信息还是明文。

2.3.2 DNS-over-Encryption

Since DNS was designed without considering security and privacy concerns, there is no encryption included in the DNS protocol, and all of DNS data are sent in plain text over the network. This vulnerability means that anyone with access to the network traffic can read this data and possibly change them for malicious purposes in a man in the middle scenario.

Several methods have been proposed to tackle this security concern, using encryption in the DNS protocol. For example, DNSCurve is a method for securing the connection between the DNS resolver and authoritative servers, or DNS-over-TLS (DoT), DNSCrypt, and DNS-over-HTTPS (DoH) are available solutions to secure the connection between the DNS clients and the DNS resolver.

2.4 DNS-over-HTTPS

DNS-over-HTTPS (or DoH) is one of the new methods of securing DNS communications that uses HTTPS protocol to wrap DNS packets. It was proposed in May 2017 by Paul Hoffman of ICANN and Patrick McManus of Mozilla and was subsequently published by IETF in October 2018 in the form of RFC8484 [22]. DoH protocol uses widely used HTTPS protocol to both secure DNS communications between DNS clients and DNS resolvers, and provide a means of resolving addresses for web applications.

To use DoH, both the client and the resolver should have the means to interpret this protocol. DoH clients wrap their DNS packets in an HTTP request with a specific media type (*application/dns-message*) and send them to the DoH server. Even though the standard may well work without encryption, It has been mentioned that this approach should only be used with HTTPS connections that are encrypted using TLS. The DoH server which acts as a DNS resolver receives this request and will respond after performing standard DNS resolving activity (such as checking DNS cache and performing DNS recursion). The response would be again in HTTPS protocol, using the media type *application/dns-message*.

DOH server有DNS resolver的作用，response的情况同样使用https协议进行包装。

Using HTTPS protocol also retains the advantages of HTTP protocol such as caching, redirection, and compression. Newer features of the HTTP protocol that are introduced in HTTP/2 can also be used in DoH for abilities that were not possible before in standard DNS connections. One of these essential features that have been mentioned in RFC8484 is HTTP/2 server push that allows DoH servers to send additional DNS records that they predict the client would need later on [22].

DOH协议本身需要
兼容之前的DNS功能，

2.4.1 Available Configurations

As backward compatibility has been one of the design requirements of DoH protocol, meaning that it should not disrupt current DNS implementations. It can be used in various configurations that help using DoH with other DNS clients and servers. There are three main deployment configurations which are detailed here:

DOH 部署的三大方式:

1、支持DOH的客户端：
eg-Firefox & Google

1. Using DoH-enabled clients: This method that is used primarily in up-to-date web browsers such as Mozilla Firefox and Google Chrome involves clients that originally use DoH instead of DNS to resolve addresses. In this scenario, a DoH client contacts the DoH server directly. For example, to enable DoH in Google Chrome browser, a user needs to type *chrome://flags/#dns-over-https* and enable *Secure DNS lookups*. Similar setting is enabled for Microsoft edge and Opera web browsers. However, for Mozilla Firefox, user needs to go to the connection settings under browser settings and enable DNS over HTTPS. In addition to enabling inbuilt DoH support for web browsers, specific operating systems and specific versions of those operating systems need to be installed to configure DoH. This is the easiest way to install a DoH in a local network.

chrome 目前默认支持DOH
且：设置-隐私和安全-Use secure DNS
在受控网络下被禁止。

2、本地系统加装DOH proxy

2. Using DoH proxies in individual hosts: In this scenario, users install a DoH proxy on their system. This proxy will capture all their DNS requests, wrap them in DoH protocol, and send them to the DoH server. It secures all of

the host's DNS communications. Figure 2.3 shows the working of the DoH protocol by installing DoH proxy in individual host.

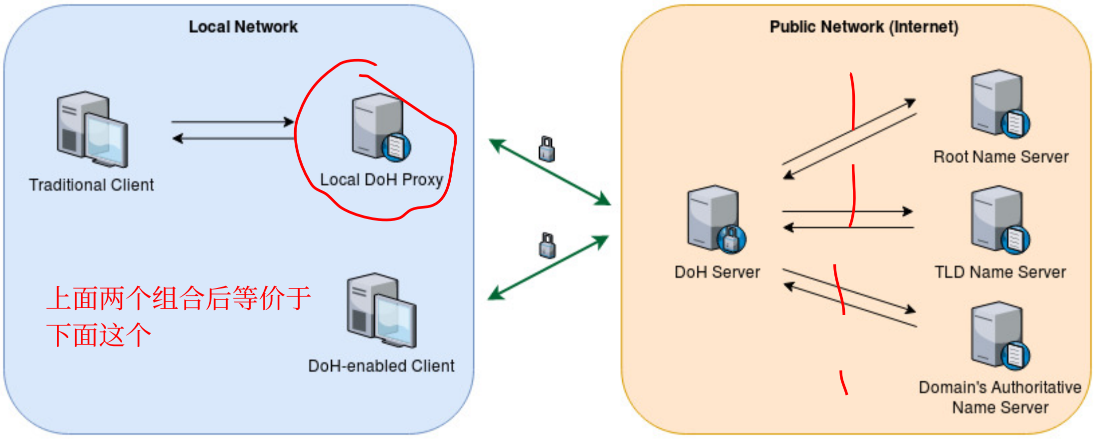


Figure 2.3: DNS-over-HTTPS by installing DoH proxy in individual host

1. A traditional client sends a DNS query to the local DoH proxy in the local network. 2, The local DoH proxy searches the response in its cache, and if it does not find it there, it encrypts that query and forwards it to DoH server over the public network. 3 The DoH server then repeatedly forwards the encrypted DNS query to a root name server, TLD name server, and domain's authoritative name server until the DoH server receives a response.Figure 2.3 also presents a DoH-enabled client that can directly send an encrypted DNS query to the DoH server. This scenario of installing DoH in a local network is challenging to implement as compared to the previous scenario but is more accessible than the next scenario.

3. Using a DoH proxy on the local name server: In this approach, the user installs DoH proxy on the local name server. DNS requests from the network are translated to the DoH and sent to the DoH server. In this configuration, the network's DNS communications can be secured, without the need to configure each one of the hosts in the network. Furthermore, this method does not need

users to be informed about the technicalities of DNS and DoH configuration.

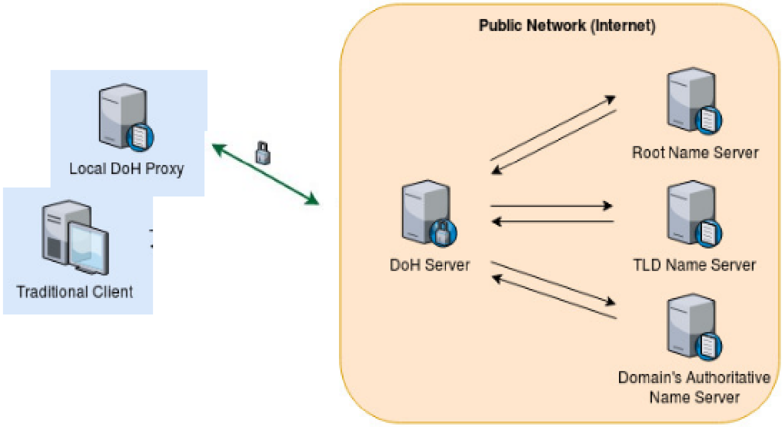


Figure 2.4: DNS-over-HTTPS by installing proxy on local name server

Figure 2.4 presents the deployment of DoH in this scenario in which the local DoH proxy is installed within a traditional client as a name server. The working of DoH remains the same as the previous scenario. This is the most challenging scenario to deploy DoH in a local network.

2.4.2 DoH Security Concerns and Remaining Vulnerabilities

While there have been many studies on applicability and performance aspects of DoH [10, 23, 56], studies on the security aspects of DoH remains limited. Two objectives seem to have taken the spotlight in DoH security: 1) Privacy and 2) Data Exfiltration. I identify and discuss major security concerns in DoH security in this sub-section ($Cont_2$ covered).

Siby et al. [52] challenge the privacy of DoH protocol, showing that it is technically possible to defer private information about one's web browsing activity by capturing and analyzing their DoH traffic [52, 53]. Other authors also studied DoH privacy aspects, with varying opinions [8, 33]. This is because while DoH encrypts the traffic between hops, mitigating man-in-the-middle attacks, the widespread use of DoH may result in the concentration of queries in a handful of DoH service providers. By collecting name resolution information, these service providers may be able to identify users and target them with advertisements or sell their data.

Data exfiltration is another issue with the DoH protocol. DNS tunnels have been widely used by malware to transfer data in and out of networks covertly [40]. Many security researchers have already criticized DoH as a new protocol for resolving domain names for making DNS tunnels harder to detect and mitigate. This will pose a great danger to enterprise networks [12, 27]. There are several ways DoH can help malware with their DNS tunnels. This list includes some of these concerns:

基础设施无法识别DNS请求，如防火墙

- **Encryption of DNS requests and responses:** Since the DoH wraps the DNS traffic in HTTPS, the DNS traffic is unavailable to the network infrastructure between the client (malware) and server (DoH server). This effectively makes detection methods that relied on examining the DNS packets obsolete for the firewalls.

之前的基于DNS请求次数作为特征的情况无法实现。

- **Hiding the number of requests and responses:** HTTP/2 is the minimum version of HTTP that DoH standard (RFC8484) recommends for using with DoH. Malware can utilize this HTTP/2 connection, to send several DoH requests, without creating a separate connection (or packet) for each request. The same also applies to the responses that DoH server is sending to the malware. Through this method, malware can hide the frequency of their DNS resolution, further reducing the number of methods that can be used to detect DNS tunnels.

- **Server push:** C2 servers may also be able to utilize ”server push”, a feature of HTTP/2 that is also part of DoH standard. Using ”server push”, servers can send DNS responses to clients, without an explicit DNS request from the clients. Also, it dependents on DoH server implementation, and might not be available using current DoH servers.

2.4.3 DNS Tunneled Traffic Characterization

Bigram 双子母组合
1、计算real domain的
bigram 访问频率。

To further focus on detecting covert channels in DoH protocol (DoH tunnels), first, the related studies on DNS tunnel detection are examined in this section. Qi et al. [44] proposed a bigram based DNS tunnel detection algorithm to define a scoring system for domain names to demonstrate the frequency of their bigrams in real domains. Their main idea is that real domain names are made by humans, and their bigrams follow Zipf's law:

词频排序定理：如果将一篇较长文章中的词频统计(递减)，并从高到低标为1,2,3...。频次最小的词等级为D，若f表示频次，r表示等级序号，则:fr = C

$$freq(i) = \frac{freq(1)}{i}$$

In other words, the frequency of any bigram is inversely correlated to its frequency rank.

1

On the other hand, since DNS tunnels use base64 and base32 to encode data, their domains have bigrams with random distribution, and thus not consistent with Zipf's law. Using this idea, they define a scoring system for domain names, that show how frequent their bigrams are, in real domains. They use a dataset of real domains, to find the frequency of each bigram, and then use it to create an online classifier that can detect DNS tunnels by scoring each DNS packet in the traffic. Their result show improvement over earlier methods that used letter frequency and reduced the false positive rate significantly.

是否可以类似抓取
网关的流量，使用
其中的DNS tunnel traffic

2

Buczak et al. [11] used traffic captured at devices of an enterprise network and traffic data at the perimeter of the network to find the best place to capture and analyze data. Their captured data contains DNS tunnel traffic from various sources. They used these datasets to train a classifier using random forests algorithm. They selected features from the previous works on this area and used part of the traffic data from known sources as their training data. Their results showed that their method is sufficiently significant, even when the classifier has not seen the tunneling

在没有明确label的情况下，能够区分出 经过隧道加密的流量数据。

technique in the training set and determined which variables and features for the random forest model work better.

Ellens et al. [17] defined a flow-based DNS tunnel traffic detection method that grouped DNS packets with similar properties (such as source, destination, protocol, etc.). Authors captured benign traffic in the network and used open-source tool to create tunneled traffic for web browsing, SSH communication and file transfer. Three categories of anomaly detectors were applied to detect anomalies in captured data without inspecting the packets deeply. Results showed that flow-based detection is promising and capable of detecting all the tunnels used in the experiment irrespective of some false-positive results.

1、以某种规则合并相似DNS packets
2、将良性流量通过开源工具构造成隧道流量：web-browsing、SSH communication、file transfer,
3、3类anomaly detector 被组合用于确认DNS packet.

Do et al. [18] worked on tunneling traffic in mobile networks emphasizing on bypass procedures used by some mobile applications to avoid firewalls. Authors analyzed the shortcomings of payload and traffic analysis methods proposed earlier. They analyzed the legitimate and malicious traffic using k-means and One Class Super Vector Machine (OCSVM). Their analysis results showed that OCSVM with some specific settings could be used to detect DNS tunnels without using predetermined features used by previous researchers.

Packet-based method of classification is gaining importance since 2019. In [57], researchers proposed a packet-based method of classification for DNS tunnel traffic by turning the DNS packets to an ASCII vector, removing features dependent on the test platform, and padding them to 400 bytes by adding zero values. Authors used ML models in the classification module, and results showed that using their feature selection method, they can achieve almost perfect precision and recall.

57：基于数据包进行检测的方法-turning DNS packets to an ASCII vector

Furthermore, Liu et al. [30] proposed a packet-based method for detecting DNS tunnels, focused on the byte-level presentation of packets in 2019. They divided each packet into 300 bytes (padding may be required), and each byte is encoded using a one-hot method into 257 values (256 plus one specifying padding bytes). They sent this representation to an embedding layer, which changed these 257 values to 64 features. Authors compared their results to standard ML algorithms and showed that their method could surpass those methods by a small margin. Their final suggested method was a CNN algorithm with these features and windows size 3, 4, and 5 sliding through an embedded representation of 300 bytes of the packet.

CNN+3,4,5 window size
+64 features.

Moreover, Mouhammd and Tariq [6] presented a thorough review of available tools explicitly developed for DNS tunneling and other tools that have other primary purposes but utilise DNS tunneling as a data exfiltration method. They illustrated the experimentation with Iodine DNS tunnels, and it shows how DNS tunneling could encapsulate other protocols such as SSH. Authors also described practical approaches for DNS tunnel detection using Snort, a popular IDS tool.

Liu et al. [31] discussed employing binary classifiers in recursive DNS servers for online DNS tunnel detection and prevention. Their proposed classifier uses 18 features constructed from the time interval between request and response, request packet size, sub-domain entropy, and record type of DNS communications. The input to the classifier is created by using a sliding window on the pairs of DNS request/response. They provided classification results for several ML trained models using this feature set and compared results for various window sizes.

binary classifier in recursive
DNS server + 18 features from time interval between request and response

Nadler et al. [38] provided a thorough history of DNS tunneling used by malware and detection algorithms that has been proposed over the last few years. They proposed architecture for detecting and blocking high-throughput and low-throughput data exfiltration over DNS. In the data collection phase, DNS queries were collected and grouped by their domain name. In the feature extraction phase, several features were computed over a sliding window, to be used in the detection of anomalous DNS requests. These features were then used in the isolation forest algorithm, which is an anomaly detector method.

This produced a model, that could be used on each sample of the data, to calculate an anomaly score that shows how much the sample differs from the legitimate traffic model. Using this method, requests to malicious domains can be detected and dropped. Their proposed method is verified using a real-world dataset from several public recursive DNS servers with a high volume of DNS traffic. The anomalous traffic used in the study was generated using several known malware and open source tools available for implementing DNS tunnels. A comparison with other works in this subject shows that their proposed method is a more robust way of detecting DNS tunnels, especially for the malware with low throughput traffics.

2.4.4 Encrypted Traffic Characterization

加密流量的分类困难程度。

A thorough analysis is carried out to provide insights into lessons learned and challenges in the classification of encrypted traffic using deep learning [5, 41, 46]. It is observed that lack of human-generated and up-to-date public datasets is a major challenge in encrypted traffic classification. In addition to that unbiased and informative input is important for feeding to the complex deep learning classifiers as they depend on the size of input for predictions. Further, it is evident that none of the deep learning classifiers is the best choice for all types of data inputs. Researchers also frequently overlook hyperparameter tuning while performing encrypted traffic characterization.

Yang et al. [59] developed a payload-based classification method to identify Transport Layer Security (TLS) traffic by analyzing TLS connection handshake packets using the Bayesian neural network. They captured the campus network traffic and collected in total 328,155 packets and categorized into web, mail, file and VoIP protocols. They labelled traffic flows using port numbers corresponding to classified applications. Three parameters in the client *Hello* message viz cipher suites, compression methods, and they used TLS extensions to extract 392 features per flow. Authors claimed that their proposed method outperformed existing payload-based

web、mail、file、VoIP 协议：32，8155 个包

TLS 连接过程中的握手协议包。

website fingerpint：用户回访网站过程尝试识别用户身份，或跨不同网站识别同一用户。

classification methods with 99% accuracy. https + website fingerprint

Zhang et al. in [60] used HTTP/2 based website fingerprinting technique to classify encrypted TLS traffic based on local request and response sequence (LRRS) feature set. They used deep forest classifier with 150 fine-grained extracted features from LRRS created from size, direction, and the number of packets. The authors divided the feature set into three parts containing 10, 80 and 60 features, respectively. Also, they created four traffic flow datasets with 12,500 page views include HTTP/1.1 and HTTP/2 protocols. They compared the results with the results obtained from six common ML algorithms namely Random Forest, SVM, Decision Tree, KNN, Logistic regression and Naïve Bayes. The authors proved that their method worked better, especially when dealing with traffic from pages other than the main page. Similarly, Houser et al. [24] developed a DoT fingerprinting method to analyze DoT traffic generated by visiting websites.

Siby et al. [52] provided a novel approach for website fingerprinting by analyzing DoH and DoT traffic. They used the patterns of encrypted DNS traffic to create fingerprints for each website in their dataset. They made a feature set out of the captured encrypted HTTPS traffic which consisted of n-gram representation of the size of packets with negative numbers indicating packets in the reverse direction. A random forest classifier was then applied to identify the website and authors have shown that with an accuracy of over 80% their novel approach can raise significant privacy concerns about current encrypted DNS practices. They modelled the temporal patterns of DNS packets and demonstrated that when DNS packets are not padded, the website fingerprinting reveals the sensitive websites visited by users. Furthermore, even if the DNS packets are padded, their method can identify DoT traffic confirming information leakage in DoT packets.

Lotfollahi et al. [32] focused on feature extraction and classification to handle network traffic characterization and identify end-user applications. Authors used

Stacked AutoEncoder and Convolutional Neural Network for network classification. They used ISCX VPN-nonVPN dataset, and the model obtained 98% accuracy for application identification and 93% accuracy for traffic characterization. Leroux et al. [29] also worked on the same dataset with ML models based on size and timing features to fingerprint VPN and ToR encrypted traffic. Authors compared their classifier with the Decision Tree (DT), Random Forest (RF), Logistic Regression (LR), and Naïve Bayes (NB) to show that SAE and CNN are better classifiers for this problem.

Patsakis et al. [42] investigated the use of Domain Generation Algorithms (DGAs) to hide malicious DNS queries in a covert channel. They used Hodrick-Prescott filter to classify traffic generated through several DGAs and studied the possibility of using DNSCurve and DNSCrypt by botnets to communicate with C&C server. Their work successfully constructed indicators of compromise (IoCs) even in the covert channel. They showcase the use of traffic analysis to provide a lightweight security mechanism and address future challenges dealing with masqueraded DNS queries.

笔记中paper1
基于HP 滤波的DNS tunnel
和 botnet 检测

A Quick UDP Internet Connection (QUIC) protocol based CNN classifier is developed by Tong et al. [55] which used flow-based and packet-based features to identify some QUIC protocol based Google services with an accuracy of approximately 99%. Their model first used a random forest classifier for differentiating low throughput services from high throughput services, then a CNN classifier for multi-class classification of video streaming, file transfer, and Google Play Music. However, the model suffered from high run-time of processing and classification due to use of flow-based features.

2.5 Summary

DoH encrypts the DNS communication between a DNS client and a DNS server to prevent eavesdropping. The encrypted communication also prevents man-in-the-middle attack by creating covert channels. DoH is still an evolving concept with many pros and cons. Since covert channels are hard to detect, DoH faces the criticism from a section of researchers. This chapter discussed the working of DoH protocol and available configuration scenarios in which DoH can be deployed in a local network. It can be concluded that none of the deployment scenarios is the best and suitable for capturing all types of network traffic.

DNS protocol is vulnerable to several attacks such as DNS forgery, covert DNS channels, DNS rebinding, network reconnaissance, and denial of service. Similarly, DoH is also exploitable by some of these attacks such as covert DNS channels. DoH also has privacy issues and is prone to data exfiltration attacks. Several attempts have been made in the past to identify covert channels so that data exfiltration initiated by the adversaries can be detected.

Although encrypted and DNS tunneled traffic characterization has received tremendous research focus as evident from review and survey papers in the past due to its implications on traffic shaping, flow control and congestion control; DoH traffic characterization is still an evolving concept. The research presented in this thesis attempts to detect and characterize DoH traffic in an online environment.

Since many of the DNS tunnel detection studies rely on the content of the DNS packets (most notably configuration of the sub-domains used in DNS requests), we can not use their methods for detecting the DoH tunnels. Furthermore, even though encrypted traffic characterization is a well-researched area, the methods used in this area of research are mostly used to fingerprint websites or detect network traces from different applications. To use these methods for DoH tunnel detection, we need to tune these methods for DoH traffic and differentiate between malicious-DoH and benign-DoH tunnel traffic. This chapter also covered $Cont_1$ and $Cont_2$, as defined is 1.2.

Chapter 3

Proposed Framework

In the previous chapter, I presented an overview of the DoH protocol and its security shortcomings based on previous and current research. One of these security issues is the covert communication through DoH protocol, otherwise called the DoH tunnel. It was shown that even though DNS tunnels have been around for a long time and there are plenty of research projects around how they can be mitigated, DoH tunnels are reasonably new and resistant to many of these detection and mitigation techniques.

doh 协议将DNS 内容加密后，基于内容的tunnel检测方法不再适用。

The main problem introduced by DoH tunnels arises from the encryption included in the DoH protocol, which prevents the use of many DNS tunnel mitigation techniques that rely on the contents of network packets to detect DNS tunnels. There is another challenge with the detection of DoH tunnels also. Unlike the DNS protocol that traditionally uses port 53 to transport the data over the network, DoH protocol works as a higher-level protocol on top of the HTTPS protocol, which uses port 443. It means while DNS packets are easily filterable on the network by their destination port and protocol, DoH packets (and connections) are harder to detect since they are virtually indistinguishable from non-DoH HTTPS traffic.

DNS流量能够从53端口直接过滤，DOH 的流量从443端口过滤困难。

This chapter details my proposed two-layered detection framework for the DoH traffic and the DoH tunnels. The proposed framework is used to classify DoH and Non-DoH traffic in the first layer and characterize DoH traffic into benign-DoH and malicious-DoH in the second layer with the detailed description of individual components, feature set used, and classification technique. The DoH characterization presents a label used to indicate that the DoH protocol is being abused to create covert communication channels.

3.1 Overview

To successfully mitigate DoH tunnels, first, we have to detect them in the network traffic. The detection framework described in this thesis consists of a two-layer architecture. Since the DoH traffic is indistinguishable from other types of HTTPS traffic, the first layer of this framework classifies the input network traffic and labels the network flows as DoH or Non-DoH.

Furthermore, layer 2 of the proposed framework characterizes those flows from the traffic that have been marked as DoH. This characterization distinguishes benign-DoH, which is the traffic created by the intended uses of DoH (Domain resolution), and malicious-DoH, which is the traffic created by the DoH tunnels that abuse the DoH protocol to create covert communication channels. Figure 3.1 described the proposed abnormal DoH traffic detection methodology in consisting of three modules: 1) Data pre-processing module, 2) Feature extraction module, and 3) Classification module. Each of these modules is further discussed in the following chapters.

To cover all parts of this framework, I designed and developed *DoHlyzer*, an automated tool written Python which is publicly available on GitHub [21]. This tool contains several modules that would help us in the implementation and analysis of the proposed framework:

1. ***DoHDataCollector*:** This module uses automation tools such as Fabric li-

虚拟机运行不同应用采集数据+tcpdump 抓取流量及分析

brary and SSH to control several virtual machines for data collection. By remotely controlling these machines, I run predefined scenarios based on our needs to collect the needed data. Each of these scenarios consists of running various tools to generate the traffic and capture them using *tcpdump*, which is network traffic capturing and analysis tool. This module corresponds with the data capture and pre-processing module of the framework, which is shown in the top box in Figure 3.1.

2. ***DoHMeter*:** Following the collection of the data by the Collector module, the Meter module extracts the necessary features from the collected traffic by utilizing the Scapy library in Python. Even though this module is used after the Collector module, it can also work independently and capture online traffic from the host for feature extraction. This module can work with two modes, one for extracting the statistical features in the form of a CSV file and the other for extracting time-series features that are saved in several JSON files. This module corresponds with the second module of the proposed framework, the feature extraction module.

3. ***DoHAnalyzer*:** This module uses the extracted time-series features from the traffic dataset, to create and test deep learning classifiers capable of binary classification of time-series input. To create such classifiers, this module utilizes Keras and TensorFlow libraries in Python. This module corresponds with the third module of the proposed framework, the classification and characterization module.

4. ***DoHVisualizer*:** This module uses Matplotlib library in Python to visualize the extracted time-series features to provide a graphical representation of the data.

step1：通过建立多个虚拟机上跑应用，获取network traffic【tcpdump】，不同类型的应用区分为不同的标记。
step2：提取time-series based features
step3：构建classifier 模型

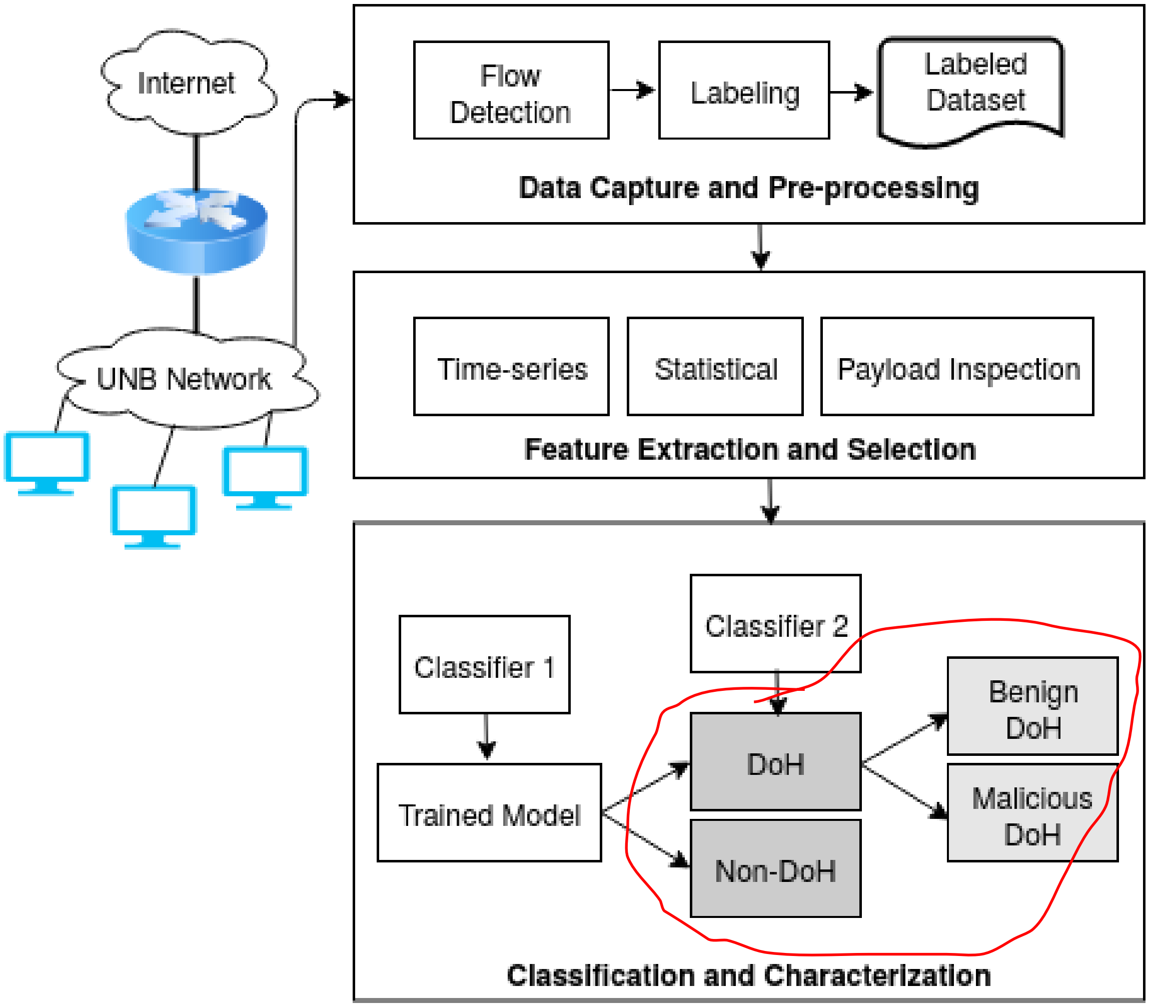


Figure 3.1: DoH traffic classification and characterization architecture

3.2 Data capture and pre-processing module

To create the necessary classifiers for this detection framework, there needs to be a dataset available to train and test the classifiers successfully. Since there has not been a great focus on DoH traffic analysis in the literature, no publicly available dataset was found that could be used for this research. To overcome this problem, a new dataset was created for this thesis. This dataset contains HTTPS traffic captured using various tools (detailed in Chapter 4).

As part of the *DoHlyzer* package, I developed the collector module in Python to help us with running these tools, generating the desired network traffic and collecting it in my dataset. Due to the need for having a comprehensive dataset which would require repeatedly running various tools with different settings, and also to isolate these experiments, I chose to run my data collection on virtual machines. The collector module manages these experiments by controlling these machines by SSH protocol using Fabric library in Python. The experiments used for data collection are defined in scenarios that the collector module interprets and runs on the machines. These scenarios indicate the tools and the settings needed for generating the data. Collection of the traffic is done by **tcpdump** and then the PCAP files are collected on the primary host (the host running the *DoHDataCollector* module).

The HTTPS traffic captured in this dataset contains all different types of traffics that I work with in this research. The dataset includes non-DoH HTTPS traffic, benign-DoH traffic created in the process of domain resolution and traffic made by DoH tunneling tools. In the next step, the data pre-processing process identifies every network traffic flow captured from the encrypted network traffic. RFC 2722, titled ”Traffic Flow Measurement: Architecture”, defines flow as “a stream of packets observed by the meter as they pass across a network between two endpoints (or from a single endpoint), which have been summarized by a traffic meter for analysis purposes”[49].

In this thesis, several values have been used to derive flows from the traffic. These values are source IP, destination IP, source port, destination port and protocol. In other words, the packets in the dataset are grouped by these values, with each packet belonging to the flow that is indicated by these values. Since the protocol (TCP) and the destination port (443) remain the same for all the flows (because the captured traffic only includes HTTPS traffic), these two values are effectively ignored.

The flows in the dataset are labeled as DoH or Non-DoH based on the destination IP address of flows. This is possible because all of the DoH networks flows included in the dataset have the destination IP address of a DoH server (a server

正常DOH：web browsing　异常：DOH tunnels 的 traffic flow

that understands the DoH protocol and accepts DoH connections). DoH flows are also distinguished by tools used for generating them. Additionally, DoH flows created by simulating web browsing activity are marked as benign-DoH while the flows captured using DoH tunnels are marked as malicious DoH. While DoH protocol may theoretically be abused by attackers in a variety of ways, I used ”malicious-DoH” label for the DoH tunneling network traffic which can be used by malicious actors to create covert channels.

3.3 Feature extraction and selection module

Feature extraction module is used to extract necessary features for classification and characterization from each flow. The major categories of features used for feature extraction, as evident from the reviewed literature, are statistical features and time-series features. *DoHMeter* reads captured traffic in PCAP format which is created by tools such as *tcpdump* or *Wireshark* and extract the features. It can also capture real-time packets from the online traffic of any of the host's network interfaces. The mode of input can be specified by `-n <iface>` for capturing online traffic (`<iface>` being the network interface used) or `-f <pcap_file>` for reading packets from a PCAP file. Online traffic captures should be interrupted manually (by pressing `Ctrl-C`).

After creating the traffic and capturing it, *DoHMeter* can produce statistical features from the captured traffic. In the other language, in this step the meter module accepts the traffic captured in the dataset in form of PCAP files as input and extracts features for each flow in the input. These features are then saved into files to be further used in the next steps. This module has two different modes for statistical and time-series features that are further explained in the following sections. I present a brief overview of these features here and provide the list of features used for this research.

3.3.1 Statistical features

Statistical features are considered powerful for performing network traffic analyses. There are several related works on this domain which have used statistical features such as duration of the flow and inter-arrival time of packets [16, 28]. Based on features used by these papers and my experiments, twenty-eight statistical features has been selected and derived from the captured network traffic as shown in Table 3.1. I define some of the typical features below:

- Coefficient of variation (CV): It is a statistical measure of the relative dispersion of data points in a data series around the mean. Mathematically, $CV = \sigma/\mu$, where σ and μ represent standard deviation and mean, respectively.

- Skew from median: Skewness from median measures the asymmetry of the probability distribution of a real-valued random variable about its median.

- Skew from mode: Skewness from median measures the asymmetry of the probability distribution of a real-valued random variable about its mode.

Based on above definitions, these three features are computed for packet length, packet time, and request/response time difference.

These statistical features are extracted from the PCAP files included in the dataset using the meter module of DoHlyzer package. This is the first modes of the meter module. This mode is activated by using `-c` switch and extracts statistical features from the input traffic. Results are saved in a CSV file, the path of which should be specified by the user. Each row in the output CSV file would specify a flow in the input traffic. The first four columns of the CSV file indicate basic information needed to identify the flow: 1) Source IP 2) Destination IP 3) Source Port 4) Destination Port. The next two columns are the start timestamp and the duration of the flow.

Table 3.1: List of statistical features obtained

Parameter	Feature
F1	Number of flow bytes sent
F2	Rate of flow bytes sent
F3	Number of flow bytes received
F4	Rate of flow bytes received
F5-F12	Packet Length (F5: Mean, F6: Median, F7: Mode, F8: Variance, F9: Standard deviation, F10: Coefficient of variation, F11: Skew from median, F12: Skew from mode)
F13-20	Packet Time (F13: Mean, F14: Median, F15: Mode, F16: Variance, F17: Standard deviation, F18: Coefficient of variation, F19: Skew from median, F20: Skew from mode)
F21-F28	Request/response time difference (F21: Mean, F22: Median, F23: Mode, F24: Variance, F25: Standard deviation, F26: Coefficient of variation, F27: Skew from median, F28: Skew from mode)

The rest of the columns contain the statistical features. This is an example for running the meter module in its first mode: `python3 dohlyzer.py -f ./input.pcap -c ./output.csv`

3.3.2 Time-series features

Time-series representation is used to model the network owing to the nature of traffic encrypted by TLS protocol in the form of a series of packets transmitted over the period. The nature of network traffic is a series of packets transmitting over time, so we can model a network flow using a time-series representation of captured traffic. Since the traffic used in this work is encrypted by TLS protocol, the payloads on the packets would not leak any useful information about the nature of underlying traffic. On the contrary, we can use other traffic shape parameters such as the packet size, packet direction, and the time difference between packets, to infer some information about the underlying traffic [60]. I keep the packets that contain TLS application data and remove insignificant packets such as ACK packets with no payload and packets too small to carry data frames. The primary step in my method is to create packet clumps to reduce the dimensionality of data.

To create these clumps I use the meter module from the *DoHlyzer* package in its second mode. This mode is activated by the `-s` switch and generate a sequence of clumps saved in JSON format. The output path in this mode should be a directory containing two subdirectories: `doh` and `ndoh`. Each flow is saved in a file indicated by source and destination addresses and ports, in the corresponding subdirectory. The contents of the file is a list of sequences. Each sequence is a list of clumps. Each clump is a JSON object including all the parameters of a clump. This is an example for running the meter module in its second mode:

-s 的输出区分数据为 doh 和 non-doh

```
#python3 dohlyzer.py -f ./input.pcap -s ./output/
```

I define a clump of packets as a sequence of one or more consecutive packets of a network flow (having the same source and destination) in the same direction to create a new and concise representation of my data as shown in Figure 3.2.

clumps of packets 定义为一个或多个连续的network flow中的包。

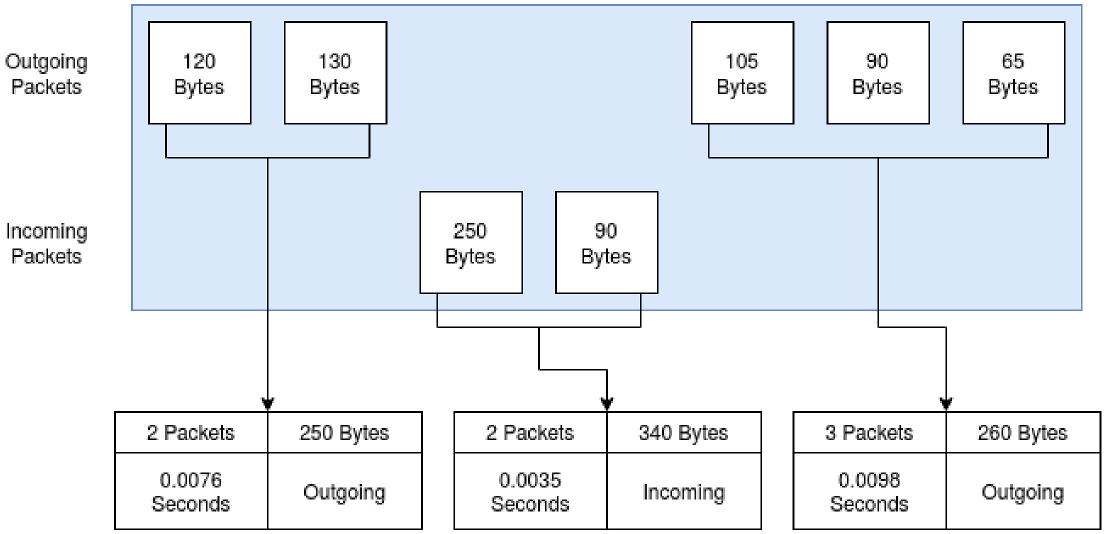


Figure 3.2: The clumping process

The rationale for this step is to combine these packets to find the application traffic scattered between several packets in the process of TLS segmentation and IP fragmentation. These aggregated data points (that I called “packet clumps” in this thesis) are sometimes referred to as bursts in the literature. A threshold timeout value for clumps is also considered so that two packets with a greater time difference do not end up in the same packet clump. Table 3.2 presents the important characteristics of clumps, that can be useful in traffic analyses:

Table 3.2: List of time-series features obtained

Parameter	Feature
F1	Size of the clump (sum of packet size in bytes)
F2	Number of packets in the clump
F3	Direction of the clump (incoming or outgoing)
F4	Duration of the clump (time difference between the first and last clump)
F5	Inter-arrival time (time difference between current and previous clump)

Each clump C is denoted by 5-tuple characteristics as:

$$C = \langle size, pktCount, direction, duration, interarrivalTime \rangle$$

By using the clumping process, a sequence of clumps for any traffic flow can be shown as

$$S = (C_1, \ldots, C_n)$$

The size of these sequences (n) depends on the network traffic inside of the flow. I use a sliding window with a length of ℓ over this sequence of clumps to generate clump sequences. Clump sequences that are smaller than the segment size are padded with empty clumps. If ℓ is a hyper-parameter that specify the number of clumps in a segment, the final feature set F_ℓ extracted from a flow is represented as:

$$F_\ell = \{(C_i, ..., C_{i+\ell}) \mid 1 \leq i < |S| - \ell\}$$

Finding the best value for ℓ is a trade-off between accuracy and response time. A smaller ℓ potentially limits the ability of the classifier to find meaningful patterns in the traffic. Thereby, it decreases the accuracy of results and time needed to capture traffic. In other words, it helps to detect and discard malicious traffic earlier. A bigger ℓ however potentially increases the accuracy and robustness of the classifier. Chapter 5 further focuses on the effects of ℓ hyperparameter on the performance of the DoH tunnel detection framework.

3.4 Classification module

I used the classification module to train the classifier and create a trained model which is used to distinguish DoH traffic from non-DoH traffic at layer 1. After getting DoH traffic at layer 1, it is characterized as benign-DoH or malicious-DoH traffic at layer 2 ($Cont_3$ covered). Since there are two types of features for each flow, I used different classifiers for each feature set. Each of these classifiers is trained on my labeled dataset, so they can be called binary classifiers. Since my dataset is fully labeled, I used supervised learning classifiers in the proposed model. I used Weka, a prominent machine learning tool, to create and analyze the statistical classifiers used in this work, and *DoHAnalyzer* which is the analyzer module from DoHlyzer package to create and benchmark the time-series classifiers[19].

3.4.1 Statistical features classifier

6类方法做统计变量区分。

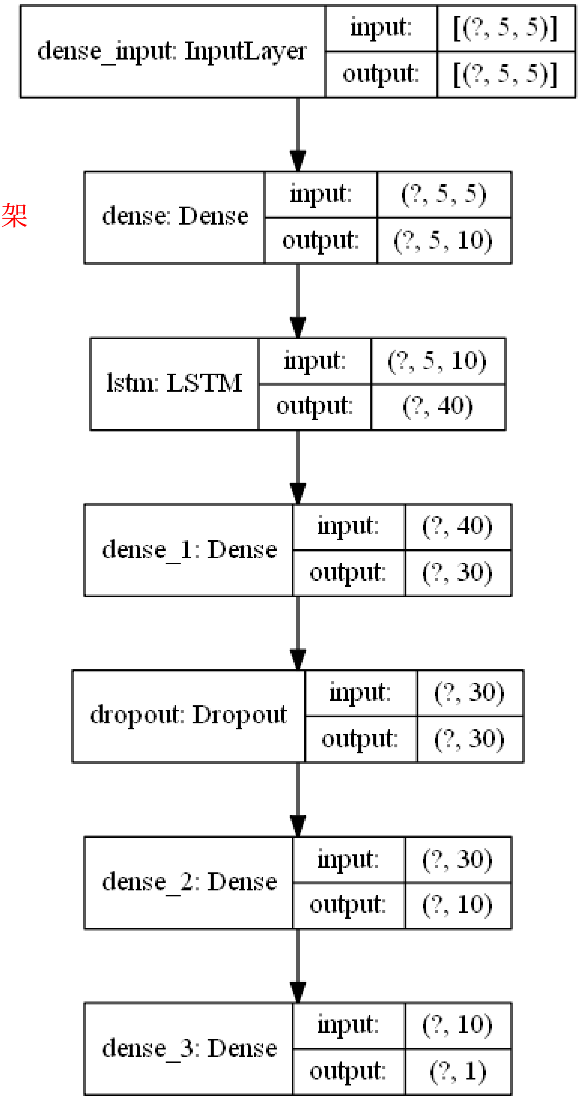
I use four common machine learning algorithms namely Random Forest (RF), Decision Tree (DT), Support Vector Machines (SVM), and Naive Bayes (NB) and two common deep learning algorithms namely deep neural network (DNN) and convolutional neural network (CNN) to classify statistical representation of flows captured at layer 1.

All these classifiers are capable of classifying encrypted traffic and have particular characteristics. RF and DT classifiers create a multitude of decision trees to train the input data and predict the class of data whereas SVM uses a statistical learning framework to support a robust prediction method. On the other hand, NB promises highly accurate results. From the deep learning classifier's perspective, DNN forms a neural network by providing a combination of input layers, hidden layers, and output layers to extract high-quality features from the raw input. Finally, CNN creates convolutions of various layers to detect complex features from input data. All these classifiers are chosen to showcase the diversity of my selection that helped to obtain the best results. The results of these classifiers are compared to obtain the best classifier for my data. It is imperative to mention here that I used *DoHMeter* to extract features from my dataset before applying statistical classifiers.

3.4.2 Time-series Classifier

For classifying time-series representations of my flows, I pre-processed the dataset to create clumps of data as explained in the previous subsection. I then used deep learning classifiers to generate a model capable of classifying sequences of clumps. Unlike the statistical features classifier that uses features calculated from the whole flow, time-series classifier can utilize a limited number of clumps to classify the flow. Recurrent neural networks such as long short-term memory (LSTM) models have been shown to work well in classifying time-series data [60]. I used a deep neural network with four hidden layers (including an LSTM layer at the second hidden layer) to create my model. 4 hidden layer (including an LSTM at 2nd hidden layer to create model)

Figure 3.3 shows all the layers of my model for an input of a sequence with 5 clumps ($\ell = 5$). The input layer of this network accepts tuples of $(\ell, clump\_size)$. Since the clumps always have 5 parameters in them, we can write this as $(\ell, 5)$. The second layer is a normal dense layer with the output of $(\ell, 10)$. The LSTM layer is the second layer and outputs a $\ell \times 8$ values. In the next dense layer this vector of values get reduced to $\ell \times 6$ and then a dropout of 0.2 is applied to the output of this layer. This dropout helps with the regularization of my model by reducing overfitting phenomenon. The last dense layer has $\ell \times 2$ nodes and is directly connected to an output node of 1. All of the layers of this model uses ReLu activation function, except for the output layer that uses a Sigmond activation function. The model is trained with an Adam optimizer with default parameters and uses a binary cross-entropy loss function.



输入层：5个clumps，每个clumps 的5个parameters

DNN time-series feature 模型框架

dense 层，输出是10维

* all layers of this model uses ReLu activation function
* output layer use Adam Boost

dropout 0.2

Figure 3.3: Plot of the DNN model with $\ell = 5$

LSTM is a special type of recurrent neural network that learns long-term dependencies. They remember information for a long period of time and easily learn to avoid long-term dependency problem. LSTMs have the ability to add or remove information from cell states regulated by logic gates. Since I created sequence of clumps to classify time-series features, LSTM turns out to be the best fit in my generated dataset to detect the number of clumps at which the best results are obtained.

I used the analyzer module in my *DoHlyzer* package for creating and analysing the LSTM classifier. This module uses Keras and Tensorflow libraries to create and train the necessary classifier for my framework. This module uses the aggregated JSON files created by the meter module that contain clumps sequences to create a LSTM model. The model is created and benchmarked using the input data. The results from the benchmark are then written in a JSON file. There are two options that should be specified: input path (using `--input` switch) and output path (using `--output` switch). This is a sample command that could be used to run this module:

```
#PYTHONPATH=../ python3 main.py --input analyzer/sample_data/
--output test.json
```

3.5 Summary

I presented the proposed framework to capture and analyze DoH and HTTPS traffic flows. This chapter elaborated the three modules of the proposed framework: (1) data capture and pre-processing, (2) feature extraction and selection, and (3) classification and characterization. This chapter also detailed the classification and characterization module of the proposed framework detects DoH traffic from non-DoH traffic at first layer and characterizes DoH traffic into Benign-DoH and Malicious-DoH traffic at second layer.

It discussed *DoHLyzer*, a tool developed in three modules to capture DoH flows, extract features and analyze them. The proposed framework collects DoH data by using *DoHDataCollector*, first module of *DoHLyzer*. Once the data is captured, *DoHMeter*, is used to extract features from the collected flows. *DoHMeter* works in two modes. The first mode of the tool extracts statistical features whilst the second mode extracts time-series features from the flows collected by *DoHDataCollector*. Finally, *DoHAnalyzer*, the third module of *DoHLyzer*, is used to analyze the DoH traffic and the last module namely *DoHVisualizer* provides a graphical representation of the data. This chapter also covered $Cont_3$.

Chapter 4

Implementation

This chapter contains the details about the implementation of the proposed framework for the detection of DoH tunnels. First, I give details about the dataset collected for this thesis. Second, I provide the implementation details of the feature extraction module based on my developed package, namely *DoHLyzer*. Appendix A shows the step by step installing and execution of *DoHLyzer*. Lastly, I explain the classification module with the details of all the different classifiers that I use in this thesis.

4.1 DoH Tunnel Dataset Collection

As mentioned in earlier chapters, I needed to create the necessary dataset for this work, since there is no publicly available dataset in the literature. The generated dataset, named *CIRA-CIC-DoHBrw-2020*, is publicly available at [3]. It consists the HTTPS traffic flows with two levels of distinct labels (The first part of $Cont_5$ covered).

这里这个数据集[3] 官网要求填写具体工作单位信息，暂时不忙获取。
PS: 找到了另一文章中的混合数据源，包含了[3]

I have DoH and Non-DoH HTTPS traffic in the first layer, and the next layer will segregate the DoH traffic as Benign-DoH and Malicious-DoH. Although malicious traffic can be tunneling as well as non-tunneling, I generated only tunneled malicious traffic for this research. I used a variety of tools to simulate network activity to

malicious traffic可以是 tunneling & non-tunneling，此处malicious 类仅使用 tunneled generate these flows. Figure 4.1 shows the overall scheme of the network model used for the dataset collection.

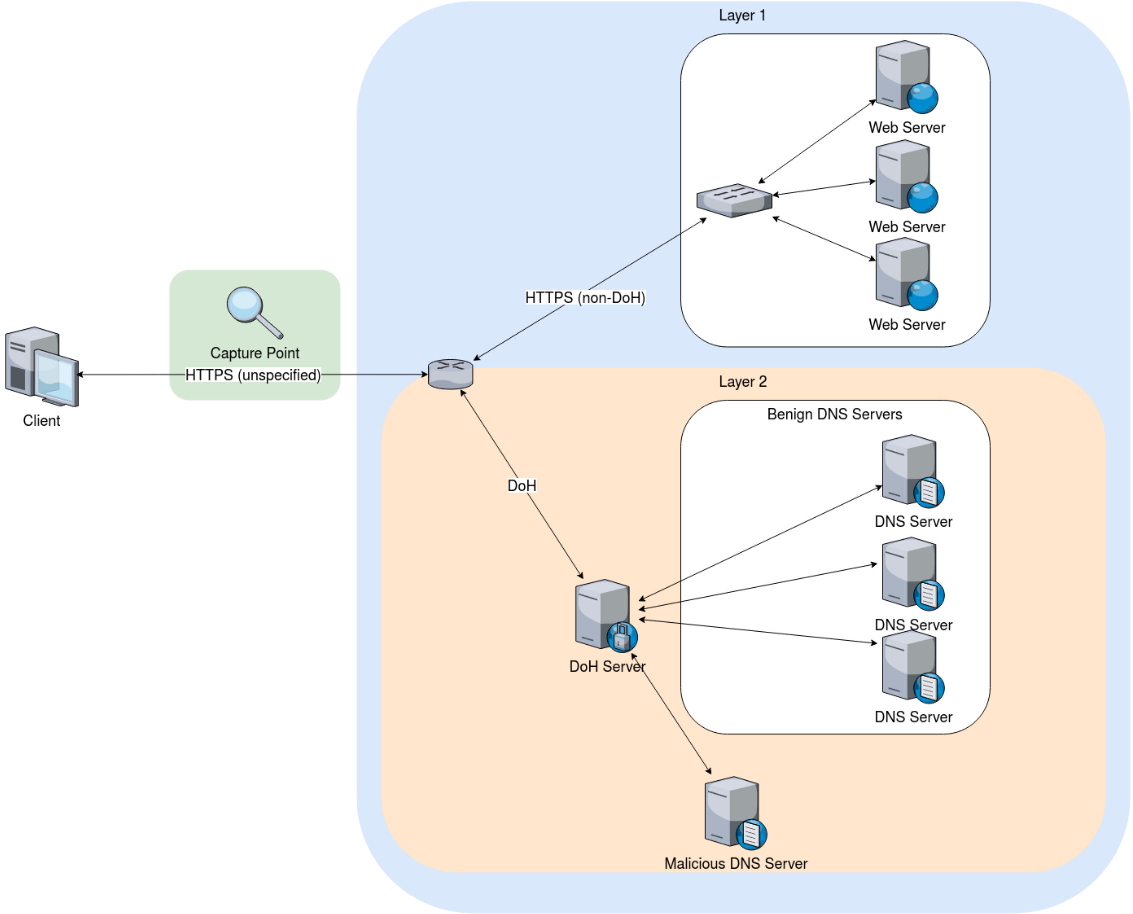


Figure 4.1: Overall view of the dataset collection network

Table 4.1 lists the most important source and destination IPs on the dataset. The public DoH IPs belong to the four public DoH providers used in the data collection: 1) AdGuard 2) Cloudflare 3) Google DNS 4) Quad9.

4.1.1 Capturing Web Browsing Network Activity

Since this dataset needs to contain all types of traffic and my DoH tunnel detection framework would encounter in its input traffic, I need to include the non-DoH HTTPS traffic also. This type of traffic can be generated by simulating web browsing activity

non-doh 通过抓包443端口的网络流量来模拟

Table 4.1: IP address used for creating the dataset

上一页提到的4个public DoH Provider 的IP Public DoH IP addresses	1.1.1.1 8.8.4.4 8.8.8.8 9.9.9.9 9.9.9.10 9.9.9.11 176.103.130.131 176.103.130.130 149.112.112.10 149.112.112.112 104.16.248.249 104.16.249.249
Source IP used to connect to websites (Google Chrome)	192.168.20.191
doh-enabled browsers, can capture benign doh Source IPs used to connect to websites (Mozilla Firefox)	192.168.20.111 192.168.20.112 192.168.20.113
几个用来模拟DOH隧道的虚拟机ip Source IPs used to create DoH tunnels	192.168.20.144 192.168.20.204 192.168.20.205 192.168.20.206 192.168.20.207 192.168.20.208 192.168.20.209 192.168.20.210 192.168.20.211 192.168.20.212

that generally uses HTTPS protocol for data transfer. By using DoH-enabled web browsers, I could capture the benign-DoH traffic that is generated during browsing sessions also.

访问 top 10k的域名模拟web browsing的过程，得到doh 和 non-doh的情况。

To simulate web browsing network activity, I captured HTTPS traffic from web browsers when visiting a series of top 10k Alexa websites [2]. Since the lazy loading of data is common in the current web ecosystem, I used simple tools, such as curl, that only load the main URL. These tools ignore the heavy parts such as the media and other requests that the page may make using AJAX and WebSockets. I used both Mozilla Firefox and Google Chrome as the two most popular web browsers. The web browsers were configured to use various public DoH resolvers to prevent biases in the dataset. My web browsing dataset generation was implemented as a

Selenium library 控制浏览器模拟网页访问过程需要安装各类驱动 https://zhuanlan.zhihu.com/p/111859925

script in Python. I used the Selenium library, a front-end testing tool, in my Python script to communicate with the web browsers.

To connect to Firefox, I also used GeckoDriver, which is an intermediary between Firefox and tools that interacts with Firefox. Since I did not need to interact graphically with the web browser, I used a headless Firefox process that does not bring up the GUI. This would help to reduce the overhead of running many web browsers in my simulations, which increased the performance of the data capturing module.

chrome 配置见 DNS_ security 笔记

Furthermore, I set up Firefox in a way that used DoH instead of DNS, so that all the name resolution generated traffics are DoH and could be captured easily. Similarly, for Google Chrome, I used chrome driver to communicate with the browser. As with Mozilla Firefox, necessary configurations were set in Google Chrome to ensure name resolution happens through DoH protocol (using various public DoH resolvers).

The classification module only works with the encrypted traffic created by HTTPS protocol which uses the destination port 443. Every flow in the captured HTTPS network traffic is analyzed, and information is collected in a tuple containing ¡source IP address, destination IP address, source port, destination port and protocol¿. Since protocol (HTTPS) and destination port (TCP 443) remains the same for all the flows, they are not considered while labeling the dataset. Therefore, every flow can only be labeled as DoH or non-DoH according to the source IP address, source port and/or destination IP address. Since the IP of DoH resolvers used in this experiment is known and those IPs are not used in other connections that are non-DoH, I can safely label every flow with a destination IP of the used DoH resolvers, as DoH traffic. All other HTTPS flows found in the traffic are labeled as non-DoH traffic. To ensure there are no outside influences on the data, all of the mentioned operations are done on VMs with no other significant HTTPS traffic. Algorithm 1 presents the capturing steps of the web browsing HTTPS traffic.

是否在p47 table 4.1 那张表里。

table 4.1 的 8个支持DOH 的ip 仅作为 doh tunnel 的web browsing 测试使用 因此，可以基于ip 区分是否doh

Algorithm 1 Data capture

```
procedure Capture($url\_file, doh\_ips$)
    for each $doh\_resolver$ in $doh\_ips$ do
        execute tcpdump
        configure browser to use $doh\_resolver$ to resolve host names
        for each $url$ in $url\_file$ do
            $t \leftarrow 5 + \text{Rand} * 5$
            open URL in browser
            Sleep($t$)
            if PCAP file size $>$ 5GB then
                break
            end if
        end for
    end for
end procedure
```

4.1.2 Capturing DoH Tunnel Network Activity

The malicious DoH part of the dataset is generated by a combination of tools used to create DoH tunnels as further detailed in this section. Traffic generated by all these tools is captured for pre-processing and training the classifiers in the next steps. To create the malicious records of my dataset, I deployed a network that can be used to simulate DoH tunneling scenarios. This is achieved by setting up a domain, setting my authoritative server as the nameserver for that domain, and using DoH requests/responses to carry data between client and server.

An essential issue in the process of capturing DoH tunnel data was finding the necessary DoH tools able to create sufficient tunneling traffic. Since the DoH protocol is backwards compatible with DNS authoritative nameservers, I used the existing

DNS tunneling tools to setup my authoritative server. For the client, I used a DoH proxy to encapsulate every DNS requests received by the client into a DoH connection with a DoH Server and relay the response it received from the DoH server to the client as DNS records. Data is captured between the DoH proxy and DoH server. This setup allowed us to simulate DoH tunnels using various DNS tunneling tools that are used in the relevant studies[34]. To label the training dataset, these simulations were done separately in an isolated network so that any traffic captured can be associated with the correct label.

The *DoHDataCollector* (The first module of *DoHLyzer*) simulates different DoH tunneling scenarios and captures the resulting HTTPS traffic. In each instance of simulation, a new DoH tunnel is made over the underlying network according to different parameters used in a scenario such as:

1. **DoH Server:** *Adguard*, *Cloudflare*, *Google*, *Quad9*
2. **DNS Tunneling Tool:** *Iodine*, *DNS2TCP*, *DNScat2*
3. **Tunneling Client and Server Configurations:** Settings such as the delay between sending requests and DNS record types used.
4. **Transmission Rate:** Random value between 100 B/s to 1100 B/s
5. **Duration**

C2 stands for Command & Control Server

To generate enough data, the clients used in the simulation were run simultaneously on ten servers, all connecting to a single C2 server posing as a DNS nameserver. A central controller written in python, deployed on another server, controls the timing of simulations. This controller reads the scenarios from a configuration file with JSON format that defines each of the scenarios and uses SSH to access the clients and the server remotely to set up the simulation. Each simulation starts by running the C2 server application (such as *Iodine*) followed by running the client

DoH proxies (which wraps DNS communications in a DoH connection) and the C2 client applications. During the simulation, a TCP connection with the specified transmission rate is tunneled through the C2 connection to generate the DoH traffic. I captured the traffic between the DoH proxy and DoH server using *tcpdump*. Working on all of the captured traffic, my *DoHMeter*[21] package was used to extract statistical and time-series features as mentioned in subsection 3.3 from the PCAP files. Table 4.2 presents the details of captured packets and generated flows for different browsers and tools on four different DoH servers. A small number of flows ($n < 50$) in captured dataset contained NaN values for some of the features, mostly because the flow did not contain enough packets to calculate those features. I deleted these flows as part of the data pre-processing/cleaning procedure. I divided the resulting dataset into the training (80%) and testing (the remaining 20%) sets, to be used with all the classifiers discussed in this thesis.

4.2 Feature Extraction

Once the data is captured by using *DoHDataCollector*, the next step is to extract features from the captured DoH flows. *DoHMeter* is developed to extract features and it works in two modes. In the first mode, it extracts statistical features whilst time-series features are extracted in the second mode. This sub-section discusses the statistical and time-series features extracted in this module.

online & offline 分别对应 time-series 和 statistical 的情况

4.2.1 Statistical Features Extraction

The feature extraction process is done for all of the PCAP files in the dataset. Since the output CSV files are numerous (one CSV for each PCAP), the final input of the classifiers are created through aggregating relevant CSV files. I created two CSV files (one for each layer) with an extra column in each file, indicating the label of the

浏览器模拟的web browsing 之外的其他协议，均为DNS tunnel的情况，视为恶意。

Table 4.2: Dataset Details

Browser/Tool	**DoH Server**	**Packets**	**Flows**	**Type**
Google Chrome	AdGuard	5609K	105141	HTTPS (Non-DoH and Benign DoH)
	Cloudflare	6117K	132552	
	Google DNS	5878K	108680	
	Quad9	10737K	199090	
Mozilla Firefox	AdGuard	4943K	50485	
	Cloudflare	4299K	90260	
	Google DNS	6413K	138422	
	Quad9	4956K	92670	
dns2tcp	AdGuard	1281K	5459	Malicious DoH
	Cloudflare	3694K	6045	
	Google DNS	28711K	17423	
	Quad9	8750K	138588	
DNSCat2	AdGuard	1301K	5369	
	Cloudflare	12346K	9230	
	Google DNS	48069K	11915	
	Quad9	19309K	9108	
Iodine	AdGuard	3938K	11336	
	Cloudflare	5932K	14110	
	Google DNS	73459K	12192	
	Quad9	22668K	8975	

data (DoH/nonDoH for layer 1 and Benign/Malicious for layer 2). The aggregation of CSV files is done using the `clump_aggregator.py` which can be found in the main directory of the DoHMeter module.

4.2.2 Time-series Features Extraction

As already mentioned, the second mode of *DoHMeter* is used to extract time-series features from the captured DoH traffic flows. *DoHMeter* extracted five time-series features which are presented in Table 3.2 in the previous chapter. Clumping process is used to create packet clumps of DoH traffic flows for effectiveness of encrypted traffic classification ($Cont_4$ covered).

4.2.2.1 Clumping process

flow_process

To implement the time-series feature mentioned in Subsection 3.3.2, I first need to pre-process the data captured in the dataset and remove the unnecessary packets. My dataset captured all the encrypted traffic sent to and received by port 443 (HTTPS) including TLS handshake packets, TCP ACK packets, TLS application data packets, and miscellaneous packets such as TCP re-transmissions and TLS alert packets. The first step to pre-process the data is to classify the flows based on their traffic patterns by keeping the unchanged values in encrypted flows like the size of the packets, direction of packets, and the time difference between packets. I filter the packets that contain TLS application data and remove insignificant packets such as HTTP/2 PING frames to analyze noise-resistant application-layer traffic. The Second step in pre-processing is to create packet clumps to reduce the dimensionality of data, as explained earlier.

1）只保留TLS layer 部分的参数做特征选取
2）TLS 中不重要的包放到 noise-resistant application layer

Algorithm 2 describes this step in detail. Through this process, the input packets get aggregated to a sequence of clumps. Each clump is essentially a list of consequtive packets in the flow that are in the same direction. To prevent unrelated packets

代码配置中 expired_update_per_flow

from getting into the same clump, there is also a timeout value to make sure the packets in a clumps are not too far apart. So while consecutive clumps usually are in the opposite direction of each other, it is possible to have consecutive clumps with the same direction in the sequence of clumps. The DoHMeter module which is responsible for extracting these clumps, outputs this sequence of clumps in a JSON file after the flow is finished through the timeout process explained in Section 4.2.1. Since there is a JSON file for each flow, the output that the DoHMeter creates from a PCAP file, would be saved in several JSON files in the output directory. Using `clump_aggregator.py` script from the DoHMeter, these JSON files can be aggregated into a single JSON file. To create the input for the next steps of the experiment (the time-series classifier), I generate these JSON files (clump sequences) for each layer by performing the feature extraction of the relevant parts of the dataset. I then aggregated these JSON files to two files for each layer (each file for one of the label). These files would later be used for the purpose of training and benchmarking the time-series classifer.

Algorithm 2 The clumping Process

```
procedure CREATE_CLUMPS($packets$)
    Initiate empty sequence $S$
    $currentClump \leftarrow$ EMPTYCLUMP
    $counter \leftarrow 0$
    while $counter < packets.length$ do
        repeat
            Append $packets[counter]$ to $currentClump$
            $counter \leftarrow counter + 1$
            $timePassed \leftarrow packets[counter].time - packets[counter-1].time$
        until $currentClump.direction \neq packets[counter].direction$ or $timePassed > timeout$
        Append $currentClump$ to $S$
        $currentClump \leftarrow$ EMPTYCLUMP
    end while
    return $S$
end procedure
```

4.2.2.2 Visualizing clump sequences

重构代码中main_visualizer.py 部分可供测试

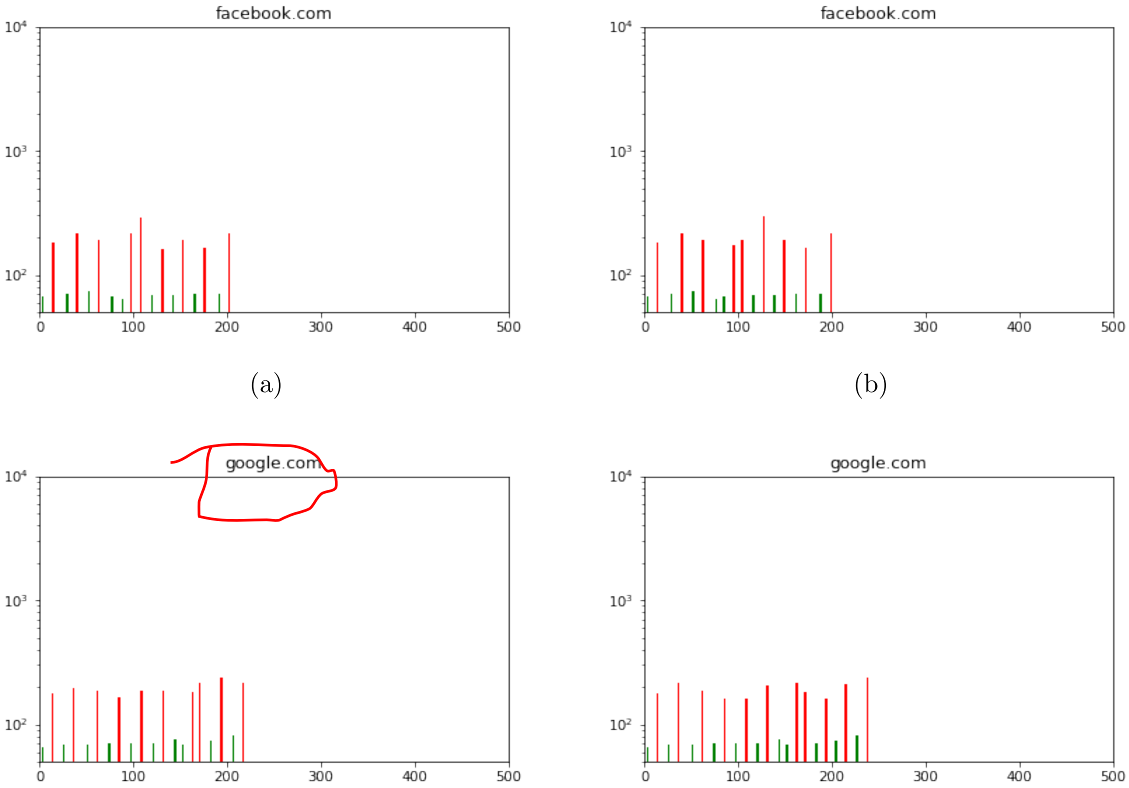
The clump sequences, each corresponding to a network flow, can be visualized to view how they can help us differentiate between different network flows. Usually, network flows are visualized by their rate of transmitting on a plot. Such visualizations are however not useful for characterizing the traffic, since they are using a linear scale axis for time. Using the linear scale axis, we are either limited to plotting a short time range (100ms) which wouldn't paint a clear picture of the pattern of a flow or plotting a large time ranges (10s) where the most of the plot is empty and the parts showing the traffic are too crowded since they are showing a lot of packets in a small area.

不基于线性时间维度，因为时间维度上大量包可能出现非常密集的情况
基于时间间隔的log，可以在近似线性时间内，plot大量packets.

The clump sequences generated as a time-series representation are visualized in Figure 4.2 for the DoH traffic created by visiting www.facebook.com and www.google.com in upper pair and lower pair respectively. Each pair shows two different visits on the same web page. The reason for the redundant visits (and plots) is to show how the visiting these web pages create slightly different clump sequences of DoH resolution but are still recognizable as being from the same web page. Green bars show an outgoing clump of packets, and red bars indicate incoming clumps. The y-axis represents the size of clumps (in bytes) in a logarithmic scale to visualize small and large clumps on a single plot. In the same fashion, since the time difference between clumps could be between milliseconds to a couple of seconds, all distances between consecutive bars are also calculated on a logarithmic scale. In other words, although the x-axis represents time and all the bars are in the right order, only distances between consecutive bars can be compared with each other.

It is visible in the figures that using this type of visualization, HTTPS requests and responses can be detected and compared for the purpose of traffic characterization. While in this thesis, these clump sequences are used for detecting and characterizing DoH traffic Figure 4.2 suggests that this method could also be used to create DoH

红色：流量进，绿色：流量出。



(a)

(b)

(c)

(d)

Figure 4.2: DoH traffic for Facebook and Google DoH Traffic

website fingerprints to detect website visits from encrypted DoH traffic captured from a user's network.

4.3 Summary

In this chapter, I discussed the implementation details of the proposed framework. my dataset was described in two parts. The first part of the dataset was created by simulating web browsing activity using a Python script controlling Google Chrome and Mozilla Firefox browser instances. These browsers were used to visit Alexa's top 10k websites, and HTTPS network traffic from these websites and also DoH traffic of the browsers were captured as non-DoH and benign-DoH parts of the dataset,

这里的tunneling tool 应该是

respectively. The second part of the dataset was created by deploying various DNS tunneling tools behind a DoH proxy to create DoH tunneling traffic. This traffic was then captured as malicious-DoH part of my dataset.

I further discussed my feature extraction method using the DoH Meter tool and how it extracts statistical and time-series features from the captured network traffic. For extracting time-series features, I used a process called clumping that involves aggregating packets by putting consecutive packets with the same direction in the same clump. *DoHMeter* tool was used to extract these features and save them as JSON files that could be later used for the purpose of training my classifiers. Visualization of the clump sequence files that were saved in JSON format was done using my *DoHVisualizer* tool that creates plots visualizing the pattern of traffic that a clump sequence indicates. This chapter also covered $Cont_4$ and $Cont_5$ (the first part).

Chapter 5

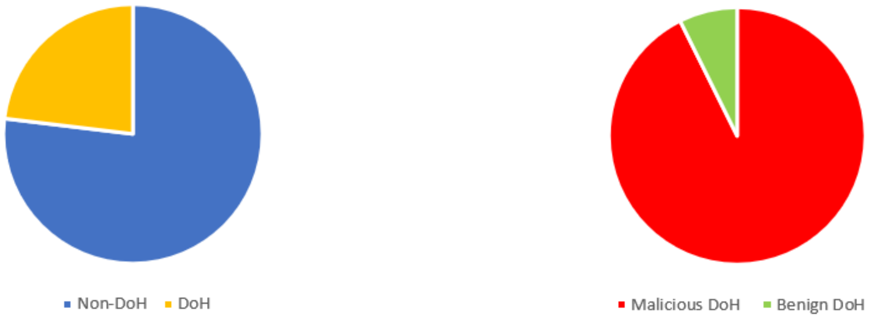
Results and Discussion

This chapter presents the generated dataset and data distribution along with the significant findings and discussion of my research by analyzing the experimentation results in both layers of the proposed model. In the analysis part, first, I focus on the results of the proposed classification technique for the first layer, which is responsible for detecting DoH flows. Secondly, I will investigate the results of the second suggested classification technique for the second layer, that characterizes DoH flows (Detected in the first layer) to two classes of benign-DoH and malicious-DoH (The second part of $Cont_5$ covered).

5.1 Data Repository and Distribution

I used my generated dataset, named *CIRA-CIC-DoHBrw-2020*, to train my classification systems and benchmarked them. As noted on Table 4.2, this dataset is a collection of PCAP files containing an overall of 1,167,050 traffic flows. Of these traffic flows, 269,557 flows are DoH (both benign and malicious), and the rest (897,492) are non-DoH. From those flows labeled as DoH, 19,807 flows are benign-DoH ones, and the other 249,750 flows are malicious-DoH (created by DoH tunnels). The two browsers used for generating non-DoH HTTPS and benign-DoH flows are Google

Chrome and Mozilla Firefox. The malicious-DoH flows were generated using three DNS tunneling tools, dns2tcp, DNSCat2, and Iodine. Figure 5.1 shows the distribution of different classes in layer 1 (chart a) and layer 2 (chart b). It is worth noting the second layer consists a subset of data in the first layer, which are the DoH flows (both the malicious and the benign DoH traffic).



(a) (b)

Figure 5.1: Distribution of different classes of traffic flows in the dataset

5.2 Layer 1: Classification of HTTPS traffic flows

My DoH tunnel detection method was based on a two-layer classification of HTTPS traffic. In the first layer, which is the focus of this section, I used all of the dataset flows as the input data for my classification. These flows are labeled as Non-DoH HTTPS or DoH. In this layer, the framework tries to detect flows that are using the DoH protocol, whether or not they are created for the purpose of DoH covert communication. This layer of framework could be used as a standalone mechanism of detecting DoH for other purposes such as blocking this protocol on an enterprise network.

layer1：检测doh流量。 Time-series

5.2.1 Classification by Statistical Features

The statistical features used in this work for classification are previously explained in Section 3.3.1. These features are derived from the data rate of the flows, packet length series, packet time series, and the series of inter-arrival time of packets (the duration between outgoing packets and the consecutive incoming packet). The complete list of these features is included in Table 3.1 from Chapter 3.

The results of the classification of HTTPS traffic at layer 1 (DoH/non-DoH classes) using statistical features are presented in Table 5.1. All the standard machine learning and deep neural network (DNN) and convolutional neural network (CNN) classifiers use statistical features calculated from the entire flow. Thus, it is essential to investigate the duration of these flows for early detection of DoH traffic by an online classifier. All the classifiers share a mean delay of 20.393 seconds to detect whether a flow can be regarded as DoH. Apparently, Random Forest (RF) and Decision Tree (DT) produced equivalent classification results with equal precision, recall and f-score value. It is followed by the support vector machine (SVM) and Naive Bayes (NB) at 0.877, 0.877 and 0.84 precision, recall and f-score value, respectively. I also evaluated deep neural network and two-dimensional (2D) CNN on the generated dataset with 0.97 and 0.98 precision and Recall respectively.

Table 5.1: DoH Traffic Classification by ML/DL

Classifier	**Precision**	**Recall**	**F-Score**	**Flow Duration (s)**	
				Mean	**Median**
RF	0.993	0.993	0.993	20.393	1.397
DT	0.993	0.993	0.993		
SVM	0.877	0.877	0.877		
NB	0.84	0.834	0.833		
DNN	0.97	0.97	0.97		
2D CNN	0.98	0.98	0.98		

5.2.2 Classification by Time-series Features

As one of the contributions of this thesis, I extracted time-series features from my data. These features are sequences of data points created by aggregating packets in the traffic flows. Since the packets in the flows used in this work contain encrypted traffic, only secondary properties of the traffic, such as packet length and packet time, are used in the creation of these features, as previously mentioned in Section 3.3.2. I introduced *packet clumps* and *clump segments* to find patterns in a limited window of traffic (instead of the whole duration of traffic flow in the case of statistical features) which in turn reduces detection latency.

I deployed long short-term memory (LSTM) a major Recurrent Neural Network (RNN) architecture to create a deep learning binary classifier as discussed in Section 3.4.2. Table 5.2 details the results of using the proposed time-series feature set in combination with LSTM architecture. It gives an ostensible picture of how the value of all evaluation metrics keeps on escalating with the number of clumps (ℓ) formed at a concise flow duration.

Table 5.2: DoH Traffic Classification by LSTM

ℓ	**Precision**	**Recall**	**F1-Score**	**Duration (s)**	
				Mean	**Median**
1	0.877	0.876	0.876	0.081	0.000
2	0.950	0.949	0.949	0.167	0.002
3	0.953	0.951	0.951	0.262	0.010
4	0.983	0.983	0.983	0.366	0.020
5	0.987	0.987	0.987	0.468	0.030
6	0.993	0.993	0.993	0.574	0.047
7	0.996	0.996	0.996	0.675	0.060
8	0.996	0.996	0.996	0.775	0.077
9	0.998	0.998	0.998	0.871	0.091
10	0.998	0.998	0.998	0.964	0.105

I consider the plot in Figure 5.2 to actuate the threshold value for the number of clumps after which DoH flows can be classified at layer 1. As we increase the number of clumps (ℓ) in the input of my time-series classifier, the precision of the results get higher. At only one clump, we can see that my classifier has a frequency of less than 88%. Using bigrams, trigrams, and n-grams of the clumps, the classifiers can use the relation between the clumps to infer the correct class. Thus, the accuracy goes up with more clumps, although this effect plateaus as ℓ gets closer to 10. It is observed that after six clumps in layer one the precision hikes beyond 0.99 offerings a precision comparable with the most accurate statistical classifiers.

threshold，6个包的时候

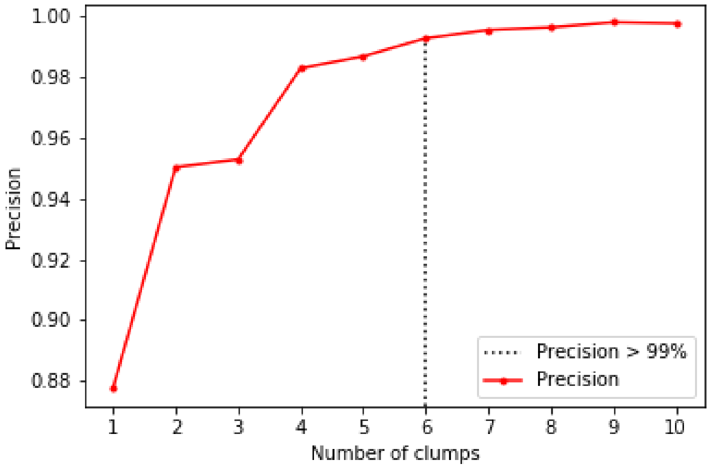
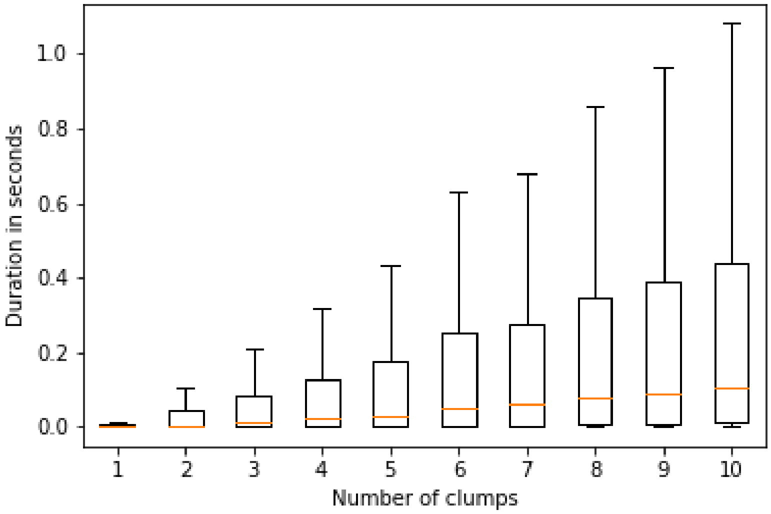


Figure 5.2: Trend of precision score per different values of ℓ in layer 1

To further analyze the effects of clumping process used in my research, I mapped out a diagram between the number of clumps and distribution of clump segment duration in the first layer in Figure 5.3. With regards to the threshold value $\ell_1 \geq 6$ for the first layer, we can derive that a DoH connection can be detected roughly under 0.8 seconds since all the segments with six clumps created from the layer one traffic have values less than 0.8 seconds (excluding the statistically insignificant outliers). Of course, for most of the flows, detection can be done even faster, since the third quartile of the data shown on the Figure 5.3 have a duration less than 0.3 seconds.



75%的数据完成检测所在的位置

Figure 5.3: Distribution of clump sequence duration in layer 1

5.3 Layer 2: Characterization of DoH traffic flows

After classifying HTTPS traffic at layer 1 to two classes of non-DoH and DoH, I further characterize the DoH traffic at layer 2 of my framework. In this layer, the implemented binary classifiers characterize the DoH traffic from layer 1 to benign DoH (which is created by normal DoH resolution of domain names) and malicious DoH (which is created by DoH tunnels for covert communication).

5.3.1 Classification by Statistical Features

The same feature set and classifiers from the Section 5.2.1 is used at layer 2 for DoH traffic characterization. The results of the characterization of DoH traffic at layer 2, to benign-DoH and malicious-DoH (DoH tunnels), using statistical features are shown in Table 5.3. Same as the previous layer, all of the ML/DL classifiers used for this layer use statistical features that are calculated on the full duration of the flow. The two most precise ML algorithms in layer 1, RF and DT algorithm, also have an excellent detection rate in layer 2, marked by an f-score of 0.999. Support Vector Machine and Naive Bayes have a mediocre f-score of 0.884 and 0.832, respectively.

The deep neural network and 2D convolutional neural network (CNN) I used also had 0.98 and 0.99 f-score, which is lower than, but comparable to the f-score of RF and DT algorithm.

The increased mean flow duration from 20.393 seconds at layer 1 to 53.924 seconds at layer 2 indicates that DoH flows generally last longer compared to other non-DoH/HTTPS flows. This is because the DoH protocol uses HTTP/2, which multiplexes requests and responses on a single long-lasting TCP connection. Because these statistical classifiers use features extracted from captured traffic of the whole duration of flows, longer lifetime of DoH flows signify the limitation of statistical features in early detection of malicious DoH traffic.

Table 5.3: DoH Characterization by ML/DL

Classifier	**Precision**	**Recall**	**F-Score**	**Flow Duration (s)**	
				Mean	**Median**
RF	0.999	0.999	0.999	53.924	34.064
DT	0.999	0.999	0.999		
SVM	0.89	0.885	0.884		
NB	0.836	0.833	0.832		
DNN	0.98	0.98	0.98		
2D CNN	0.99	0.99	0.99		

5.3.2 Classification by Time-series Features

The same time-series feature set and LSTM classifiers used in Section 5.2.2 is used at layer 2 to distinguish between malicious-DoH (generated by tunneling activity) and benign-DoH. The results from DoH traffic characterization using my time-series classifier are shown in Table 5.4.

To find an appropriate value for my hyper-parameter ℓ, the number of clumps in the input, I plotted precision value per ℓ in Figure 5.4.

Table 5.4: DoH Characterization by LSTM

ℓ	**Precision**	**Recall**	**F1-Score**	**Duration (s)**	
				Mean	**Median**
1	0.825	0.792	0.782	0.164	0.002
2	0.985	0.984	0.985	0.329	0.022
3	0.991	0.991	0.991	0.502	0.05
4	0.995	0.995	0.995	0.685	0.094
5	0.997	0.997	0.997	0.872	0.142
6	0.998	0.998	0.998	1.063	0.203
7	0.997	0.997	0.997	1.26	0.258
8	0.999	0.999	0.999	1.45	0.313
9	0.999	0.999	0.999	1.63	0.373
10	0.999	0.999	0.999	1.803	0.44

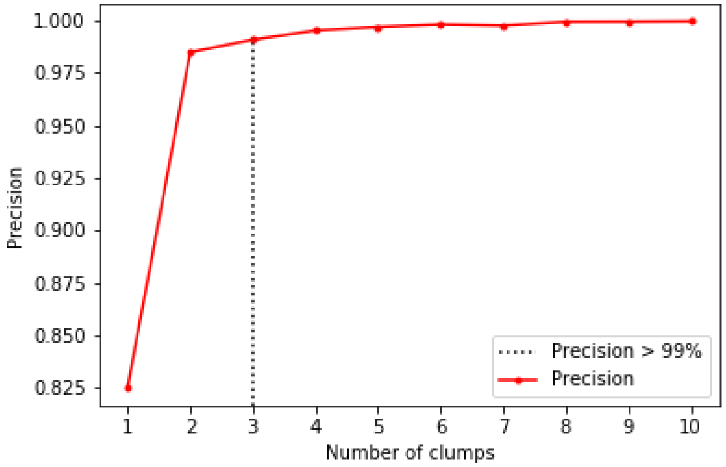


Figure 5.4: Trend of precision score per different values of ℓ in layer 2

Time-series numbers of clumps >=3

To have a precision comparable to the best of statistical classifiers, we need to have $\ell \geq 3$. Considering the distribution of clump segment duration in layer2, which is shown in Figure 5.5, such a value for ℓ would mean that my characterization algorithm could detect at least 99% of DoH tunnels in less than 1 second with only three clumps from the flow.

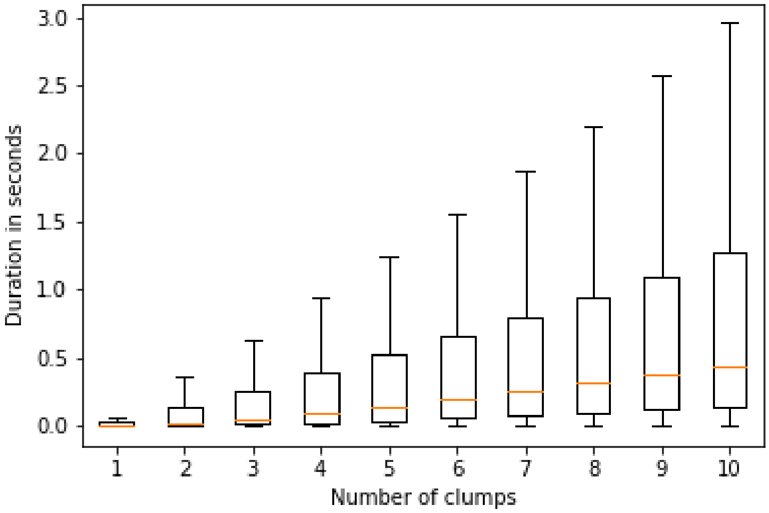


Figure 5.5: Distribution of clump sequence duration in layer 2

5.4 Summary

In this chapter, I reviewed the contents of my dataset and the distribution of different classes in each of the two layers of my framework. I also examined the classification results from both of the layers when using statistical and time-series classification. In the first layer, which is a classification of HTTPS traffic flows into Non-DoH and DoH classes, I investigated the results of various classifiers using both statistical and time-series feature sets. In the results of machine learning algorithms using my statistical features, I found random Forest and decision tree algorithms to provide the best accuracy for DoH traffic detection with a precision of more than 99%. My time-series classifier, which uses the time-series data, also performed well, when the number of clumps (ℓ) in the input was more than 6, having a precision of more than 99%. Plus, unlike the statistical features that are calculated from the whole flow, my time-series classifier uses less than 0.8 seconds of traffic to detect DoH flows, most of the time.

In the second layer, characterization of DoH traffic, again the Random forest and

C4.5 algorithms performed best with $> 99\%$ precision in detecting malicious DoH flows. My time-series classifier also performed best, when the number of clumps (ℓ) was more than 3. Here the detection was achieved by the time-series classifier in less than 1 second.

Both statistical and time-series classifiers give excellent results at both layers until we bring into consideration the fact the nature of DNS request-response pairs which are extremely short to detect encrypted DoH traffic. My proposed time-series classifier can classify and characterize DoH flows with the same precision as previously studied statistical classifiers, while outperforming them in the delay before detection (less than 1 second). Such a low delay is especially crucial in detection and prevention systems that deal with online traffic. This shows that my proposed time-series feature set and the classifier can be successfully used in online environments to detect malicious DoH traffic created by DoH tunnels. This chapter covered the second part of $Cont_5$.

Chapter 6

Conclusions and Future Work

Domain Name System (DNS) is one of the most important protocols of the Internet that has been widely used since its creation. Over the years, many security vulnerabilities has been found in the DNS protocol that prompted creation of extensions and new protocol that would help make DNS more secure. DNS-over-HTTPS (DoH) protocol is one of these efforts that help make DNS more private and fix some of the security issues by encrypting the DNS packets through the HTTPS protocol. While DoH has been praised for its ease of use and the security improvements it introduces, it hasn't been comprehensively studied to indicate how much it can help with the current vulnerabilites of DNS and what new vulnerabilites are there that needs to be studied.

In this thesis, I did a systematic study of the DNS security vulnerabilities and created a taxonomy of possible DNS attacks. Using this taxonomy I studied the effects of DoH on DNS security and analyzed the security aspects of DoH protocol. One of the most important security concerns regarding DoH protocol is covert communications through DoH. These kinds of communications in DNS protocol are called DNS tunnels and work by encoding data in DNS requests and responses. DoH protocol makes detection of DNS tunnels a concern since current DNS tunnel detection methods usually rely on deep packet inspection strategies that are impossible when dealing with the encryption in DoH protocol.

I presented the exploitation of DNS protocol to create covert channels by tunneling data through DoH connections. DoH was successfully deployed through web browsers and DNS tunneling tools to generate and capture benign and malicious DoH flows along with encrypted traffic. I created a dataset with more than 1 million records (traffic flows) by capturing the network activity generated through these scenarios. My dataset contains non-DoH HTTPS traffic, benign-DoH traffic created by DoH domain resolution and malicious-DoH traffic created by DoH tunnels. I then used statistical features to detect DoH connections and malicious-DoH activity (DoH tunneling) in a two-layered binary classification approach where the first layer distinguishes DoH traffic from non-DoH traffic and the second layer characterizes malicious and benign DoH flows. I demonstrated that malicious-DoH traffic can be accurately detected by using the proposed two-layer binary classification architecture with a precision of more than 99% when using machine learning algorithms such as Decision Tree (DT) and Random Forest (RF).

Moreover, I dealt with latency as a significant constraint when analyzing DoH request-response flows. I showed that my packet aggregation method called the clumping process facilitated early analysis of traffic allowing to create time-series classifiers capable of timely detection of DoH traffic while retaining the accuracy of statistical classifiers with more than 99% precision when using sequences of more than 6 clumps. My time-series classifier was shown to be able to detect DoH flows in at least 99% of instances in less than 0.8 seconds. My time-series classifier were also capable of detecting DoH tunnels with more that 99% precision in under 1 seconds in 99% of the instances with using only 3 clumps of the traffic flow.

6.1 Future Work

Based on the listed challenges in Section two and some limitation in my implementation, these are my future works:

- DNS padding is one of the extensions of DNS protocol that allows adding various amounts of padding to DNS packets. This padding may affects the accuracy of my framework, since it will change the size of DoH packets if DoH flows. Adversaries using DoH tunnels may use different padding strategies that can help them disguise their traffic as benign-DoH or even hide the fact that they are using DoH protocol.

- The DoH tunnel network traffic used in this work were created by passing DNS tunnel traffic through a DoH proxy. This type of traffic may have limitations compared to DoH tunnels that were created by tools that are capable of using DoH protocol themselves. For example, such a tool may use more than 1 connection or control the shape of the network traffic by introducing delays to create more unpredictable traffic patterns that are harder to detect.

- My clumping process and the resulting time-series classifiers could be used to investigate information leakage in DoH protocol. An example of this phenomenon would be detecting web page visits using the DoH fingerprint that web site create by loading resources from various domains that would lead to a predictable sequence of DNS lookups that could be detected in DoH traffic.

- Other DNS-over-Encryption protocols such as DoT (DNS-over-TLS) are not studied in this thesis. These protocols may have advantages and disadvantages in comparison with DoH in regards to DNS tunnels. The effectiveness of my two-layer detection framework when dealing with other protocols could be further investigated.

- There might be some other areas of DoH protocol that still inherit DNS vulnerabilities. Various DNS flood attacks may still occur in DoH infrastructure where detection is harder due to encryption. Also the computational overhead of DoH protocol may introduce additional challenges to circumventing these attacks.

Bibliography

[1] A New Needle and Haystack: Detecting DNS over HTTPS Usage, https://www.sans.org/reading-room/whitepapers/dns/needle-haystack-detecting-dns-https-usage-39160, Last accessed April 27, 2020.

[2] Alexa, Last accessed December 10, 2019.

[3] Mohammadreza MontazeriShatoori , Logan Davidson, Gurdip Kaur, and Arash Habibi Laskhari, CIRA-CIC-DoHBrw-2020, 2020.

[4] Donald E. Eastlake 3rd, Domain Name System Security Extensions, RFC 2535, March 1999.

[5] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescap é, Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges, IEEE Transactions on Network and Service Management **16** (2019), no. 2, 445–458.

[6] Mouhammd Al-kasassbeh and Tariq Khairallah, Winning tactics with DNS tunnelling, Network Security **2019** (2019), no. 12, 12–19.

[7] Derek Atkins and Rob Austein, Threat analysis of the domain name system (DNS), RFC 3833, August 2004.

[8] Kevin Borgolte, Tithi Chattopadhyay, Nick Feamster, Mihir Kshirsagar, Jordan Holland, Austin Hounsel, and Paul Schmitt, How DNS over HTTPS is reshaping privacy, performance, and policy in the internet ecosystem, Performance, and Policy in the Internet Ecosystem (July 27, 2019) (2019).

[9] Kenton Born and David Gustafson, Detecting DNS tunnels using character frequency analysis, arXiv preprint arXiv:1004.4358 (2010).

[10] Timm Böttger, Felix Cuadrado, Gianni Antichi, Eder Leão Fernandes, Gareth Tyson, Ignacio Castro, and Steve Uhlig, An Empirical Study of the Cost of DNS-over-HTTPS, Proceedings of the Internet Measurement Conference, 2019, pp. 15–21.

[11] Anna L Buczak, Paul A Hanke, George J Cancro, Michael K Toma, Lanier A Watkins, and Jeffrey S Chavis, Detection of tunnels in PCAP data by random forests, Proceedings of the 11th Annual Cyber and Information Security Research Conference, 2016, pp. 1–4.

[12] Kimo Bumanglag and Houssain Kettani, On the impact of DNS Over HTTPS paradigm on cyber systems, 2020 3rd International Conference on Information and Computer Technologies (ICICT), IEEE, 2020, pp. 494–499.

[13] Sebastiano Di Paola and Dario Lombardo, Protecting against DNS reflection attacks with Bloom filters, International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 2011, pp. 1–16.

[14] Christian J Dietrich, Christian Rossow, Felix C Freiling, Herbert Bos, Maarten Van Steen, and Norbert Pohlmann, On Botnets that use DNS for Command and Control, 2011 seventh european conference on computer network defense, IEEE, 2011, pp. 9–16.

[15] Michael Dooley and Timothy Rooney, DNS Security Management, John Wiley & Sons, 2017.

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

[17] Wendy Ellens, Piotr Żuraniewski, Anna Sperotto, Harm Schotanus, Michel Mandjes, and Erik Meeuwissen, Flow-based detection of DNS tunnels, IFIP International Conference on Autonomous Infrastructure, Management and Security, Springer, 2013, pp. 124–135.

[18] Paal Engelstad, Boning Feng, Thanh van Do, et al., Detection of DNS tunneling in mobile networks using machine learning, International Conference on Information Science and Applications, Springer, 2017, pp. 221–230.

[19] E. Frank, M. A. Hall, G. Holmes, R. Kirkby, B. Pfahringer, and I. H. Witten, Weka: A machine learning workbench for data mining., pp. 1305–1314, Springer, 2005.

[20] Steve Friedl, An Illustrated Guide to the Kaminsky DNS Vulnerability, `http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html`, 2008.

[21] GitHub, DoHMeter, `https://github.com/ahlashkari/DOHlyzer/tree/master/DoHMeter`, 2019.

[22] Paul E. Hoffman and Patrick McManus , DNS Queries over HTTPS (DoH) , RFC 8484, October 2018.

[23] Austin Hounsel, Kevin Borgolte, Paul Schmitt, Jordan Holland, and Nick Feamster, Analyzing the costs (and benefits) of DNS, DoT, and DoH for the modern

web, Proceedings of the Applied Networking Research Workshop, 2019, pp. 20–22.

[24] Rebekah Houser, Zhou Li, Chase Cotton, and Haining Wang, An investigation on information leakage of DNS over TLS, Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies, 2019, pp. 123–137.

[25] Zi Hu, Liang Zhu, John Heidemann, Allison Mankin, Duane Wessels, and Paul E. Hoffman, Specification for DNS over Transport Layer Security (TLS) , RFC 7858, May 2016.

[26] David Huistra, Detecting reflection attacks in DNS flows, 19th Twente Student Conference on IT, 2013.

[27] Warren Kumari, Barry Leiba, Suzanne Woolf, Joe Abley, Tim April, Paul Ebersman, Ondrej Filip, Geoff Huston, Jacques Latour, John Levine, et al., The Implications of DNS over HTTPS and DNS over TLS, (2020).

[28] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Saiful Islam Mamun, and Ali A Ghorbani, Characterization of tor traffic using time based features, ICISSP 2017 - Proc. 3rd Int. Conf. Inf. Syst. Secur. Priv., vol. 2017-Janua, SCITEPRESS - Science and Technology Publications, 2017, pp. 253–262.

[29] Sam Leroux, Steven Bohez, Pieter-Jan Maenhaut, Nathan Meheus, Pieter Simoens, and Bart Dhoedt, Fingerprinting encrypted network traffic types using machine learning, NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2018, pp. 1–5.

[30] Chang Liu, Liang Dai, Wenjing Cui, and Tao Lin, A Byte-level CNN Method to Detect DNS Tunnels, 2019 IEEE 38th Interna-

tional Performance Computing and Communications Conference (IPCCC), IEEE, 2019, pp. 1–8.

[31] Jingkun Liu, Shuhao Li, Yongzheng Zhang, Jun Xiao, Peng Chang, and Chengwei Peng, Detecting DNS tunnel through binary-classification based on behavior features, 2017 IEEE Trustcom/BigDataSE/ICESS, IEEE, 2017, pp. 339–346.

[32] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian, Deep packet: A novel approach for encrypted traffic classification using deep learning, Soft Computing **24** (2020), no. 3, 1999–2012.

[33] Chaoyi Lu, Baojun Liu, Zhou Li, Shuang Hao, Haixin Duan, Mingming Zhang, Chunying Leng, Ying Liu, Zaifeng Zhang, and Jianping Wu, An end-to-end, large-scale measurement of DNS-over-Encryption: How far have we come?, Proceedings of the Internet Measurement Conference, 2019, pp. 22–35.

[34] Alessio Merlo, Gianluca Papaleo, Stefano Veneziano, and Maurizio Aiello, A comparative performance evaluation of DNS tunneling tools, Computational Intelligence in Security for Information Systems, Springer, 2011, pp. 84–91.

[35] Roger Meyer, Impact of DNS over HTTPS (DoH) on DNS Rebinding Attacks, `https://research.nccgroup.com/2020/03/30/impact-of-dns-over-https-doh-on-dns-rebinding-attacks/`, 2020.

[36] Paul Mockapetris, Domain names - concepts and facilities , RFC 1034, November 1987.

[37] Cathal Mullaney, Morto worm sets a (DNS) record, `http://www.symantec.com/connect/blogs/morto-worm-sets-dns-record`, 2011.

[38] Asaf Nadler, Avi Aminov, and Asaf Shabtai, Detection of malicious and low throughput data exfiltration over the DNS protocol, Computers & Security **80** (2019), 36–53.

[39] Nccgroup, Singularity of Origin, `https://github.com/nccgroup/singularity`, 2020.

[40] Lucas Nussbaum, Pierre Neyron, and Olivier Richard, On robust covert channels inside DNS, IFIP International Information Security Conference, Springer, 2009, pp. 51–62.

[41] Fannia Pacheco, Ernesto Exposito, Mathieu Gineste, Cedric Baudoin, and Jose Aguilar, Towards the deployment of machine learning solutions in network traffic classification: A systematic survey, IEEE Communications Surveys & Tutorials **21** (2018), no. 2, 1988–2014.

[42] Constantinos Patsakis, Fran Casino, and Vasilios Katos, Encrypted and covert DNS queries for botnets: Challenges and countermeasures, Computers & Security **88** (2020), 101614.

[43] Nicholas A. Plante, Practical domain name system security: A survey of common hazards and preventative measures, `http://www.infosecwriters.com/text_resources/pdf/dns-security-survey.pdf`, pp. 1–20.

[44] Cheng Qi, Xiaojun Chen, Cui Xu, Jinqiao Shi, and Peipeng Liu, A bigram based real time DNS tunnel detection approach, Procedia Computer Science **17** (2013), 852–860.

[45] Daan Raman, Bjorn De Sutter, Bart Coppens, Stijn Volckaert, Koen De Bosschere, Pieter Danhieux, and Erik Van Buggenhout, DNS tunneling for network penetration, International Conference on Information Security and Cryptology, Springer, 2012, pp. 65–77.

[46] Shahbaz Rezaei and Xin Liu, Deep learning for encrypted traffic classification: An overview, IEEE communications magazine **57** (2019), no. 5, 76–81.

[47] Scott Rose, Matt Larson, Dan Massey, Rob Austein, and Roy Arends, DNS Security Introduction and Requirements , RFC 4033, March 2005.

[48] Thijs Rozekrans, Matthijs Mekking, and Javy de Koning, Defending against DNS reflection amplification attacks, University of Amsterdam System & Network Engineering RP1 (2013).

[49] Dr. Greg R. Ruth, Nevil Brownlee, and Cynthia G. Mills, Traffic Flow Measurement: Architecture, RFC 2722, October 1999.

[50] Lior Shafir, Yehuda Afek, and Anat Bremler-Barr, NXNSAttack: Recursive DNS Inefficiencies and Vulnerabilities, arXiv preprint arXiv:2005.09107 (2020).

[51] Stephen Sheridan and Anthony Keane, Detection of DNS Based Covert Channels, In Proceedings of the 14th European Conference on Cyber Warfare and Security (ECCWS), University of Hertfordshire, Hatfield, UK (2015), 1–9.

[52] Sandra Siby, Marc Juarez, Claudia Diaz, Narseo Vallina-Rodriguez, and Carmela Troncoso, Encrypted DNS–> privacy? a traffic analysis perspective, arXiv preprint arXiv:1906.09682 (2019).

[53] Sandra Siby, Marc Juarez, Narseo Vallina-Rodriguez, and Carmela Troncoso, DNS Privacy not so private: the traffic analysis perspective, (2018).

[54] Sooel Son and Vitaly Shmatikov, The hitchhiker's guide to DNS cache poisoning, International Conference on Security and Privacy in Communication Systems, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer, Berlin, Heidelberg **50** (2010), 466–483.

[55] Van Tong, Hai Anh Tran, Sami Souihi, and Abdelhamid Mellouk, A novel QUIC traffic classifier based on convolutional neural networks, 2018 IEEE Global Communications Conference (GLOBECOM), IEEE, 2018, pp. 1–6.

[56] Jeroen Wijenbergh, Veelasha Moonsamy, Roland van Rijsdijk-Deij, and DWC Daniël Kuijsters, Performance comparison of DNS over HTTPS to unencrypted DNS, (2019).

[57] First Kemeng Wu, Second Yongzheng Zhang, and Third Tao Yin, CLR: A Classification of DNS Tunnel Based on Logistic Regression, 2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC), IEEE, 2019, pp. 1–1.

[58] Kui Xu, Patrick Butler, Sudip Saha, and Danfeng Yao, DNS for massive-scale command and control, IEEE Transactions on Dependable and Secure Computing **10** (2013), no. 3, 143–153.

[59] Jiwon Yang, Jargalsaikhan Narantuya, and Hyuk Lim, Bayesian neural network based encrypted traffic classification using initial handshake packets, 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks–Supplemental Volume (DSN-S), IEEE, 2019, pp. 19–20.

[60] Ziqing Zhang, Cuicui Kang, Gang Xiong, and Zhen Li, Deep forest with LRRS feature for fine-grained website fingerprinting with encrypted SSL/TLS, Proceedings of the 28th ACM International Conference on Information and Knowledge Management, 2019, pp. 851–860.

Appendix A

Using DoHlyzer package

A.1 Requirements

This appendix offers the necessary steps to download and use the DoHlyzer package on Ubuntu and Ubuntu-based OSes.

To download and run this package, we first need to install some prerequisite packages:

```
# sudo apt install python3-venv python3-pip git
```

To download the DoHlyzer package you'd need to access the DoHlyzer repo on GitHub. You could download the package by cloning it using git:

```
# git clone https://github.com/ahlashkari/DoHlyzer.git
# cd DoHlyzer
```

Now we need to create a virtual environment for Python using venv and install the necessary Python packages, that are listed in `requirements.txt`, in there:

```
# python3 -m venv .venv
# .  .venv/bin/activate
# pip3 install -r requirements.txt
```

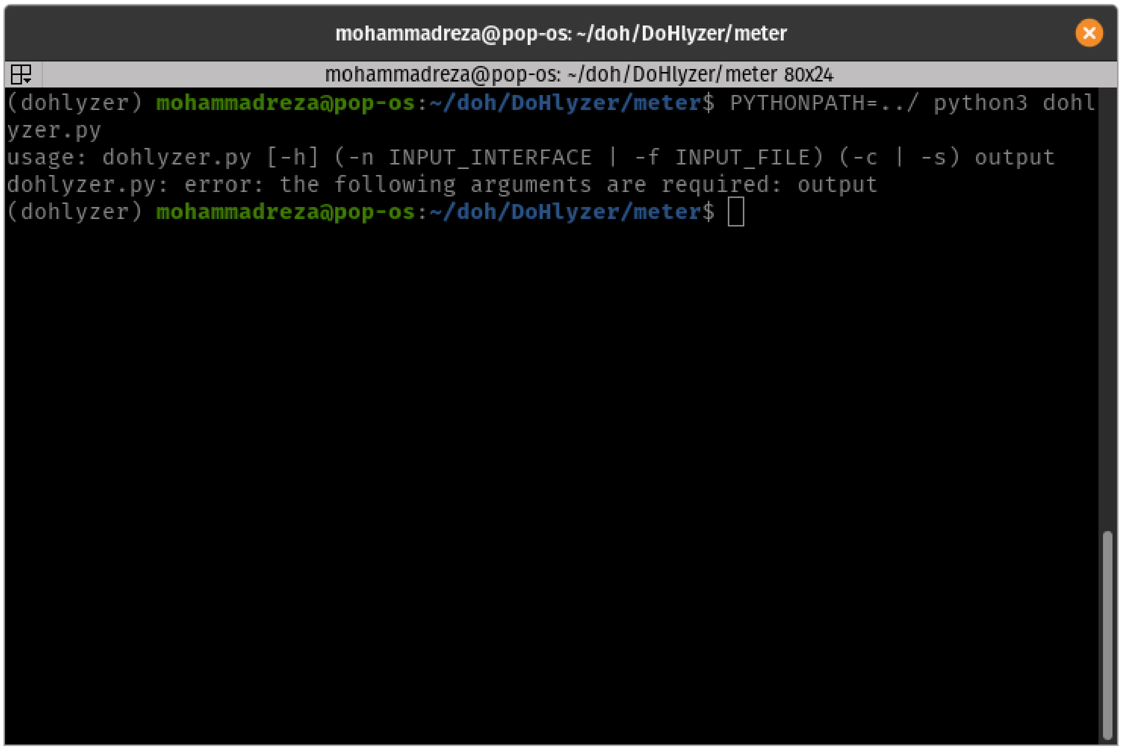
A.2 DoH Meter Module

To use the meter module, we used a test file named `test.pcap` containing both DoH and non-DoH network traffic.

```
# cd meter

# tcpdump -w test.pcap # If you want to use tcpdump to create a test PCAP

# PYTHONPATH=..  python3 dohlyzer.py # This command will result in an error
```



mohammadreza@pop-os: ~/doh/DoHlyzer/meter
mohammadreza@pop-os: ~/doh/DoHlyzer/meter 80x24

```
(dohlyzer) mohammadreza@pop-os:~/doh/DoHlyzer/meter$ PYTHONPATH=../ python3 dohl
yzer.py
usage: dohlyzer.py [-h] (-n INPUT_INTERFACE | -f INPUT_FILE) (-c | -s) output
dohlyzer.py: error: the following arguments are required: output
(dohlyzer) mohammadreza@pop-os:~/doh/DoHlyzer/meter$ 
```

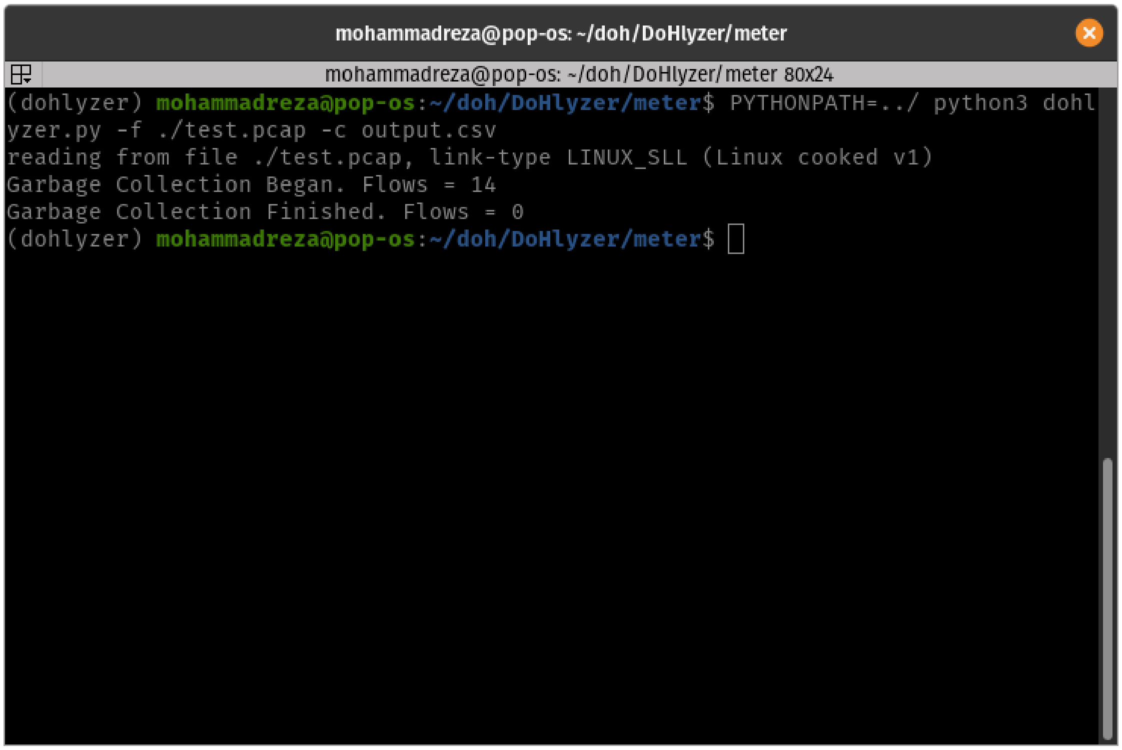
Figure A.1: Running meter module to see the help text

As you can see in Figure A.1, we need to indicate a few options for running meter module.

A.2.1 Extracting Statistical Features

Here we use `-c` switch to extract statistical features. We can see this in Figure A.2:

```
# PYTHONPATH=..  python3 dohlyzer.py -f ./test.pcap -c output.csv
```

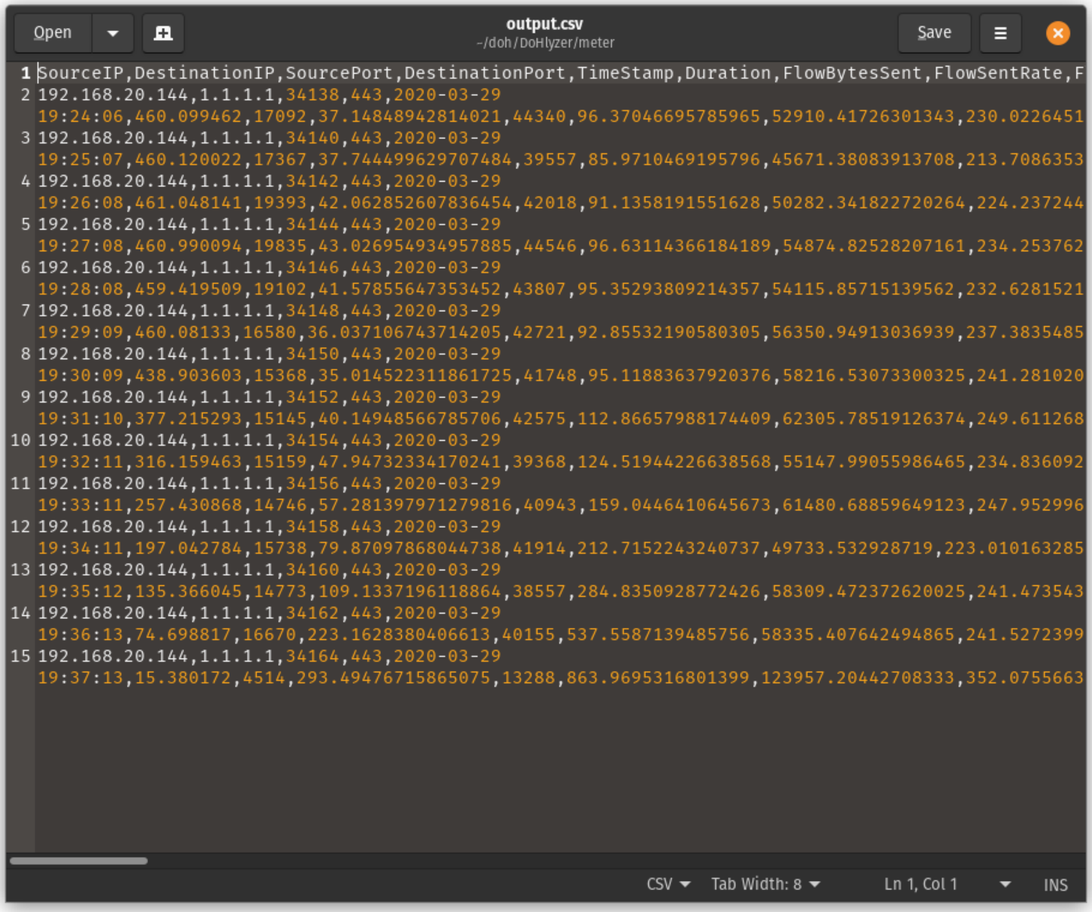


mohammadreza@pop-os: ~/doh/DoHlyzer/meter
mohammadreza@pop-os: ~/doh/DoHlyzer/meter 80x24

```
(dohlyzer) mohammadreza@pop-os:~/doh/DoHlyzer/meter$ PYTHONPATH=../ python3 dohl
yzer.py -f ./test.pcap -c output.csv
reading from file ./test.pcap, link-type LINUX_SLL (Linux cooked v1)
Garbage Collection Began. Flows = 14
Garbage Collection Finished. Flows = 0
(dohlyzer) mohammadreza@pop-os:~/doh/DoHlyzer/meter$
```

Figure A.2: Running meter module to extract statistical features

The statistical features are saved in `output.csv` as we indicated in the command (See Figure A.3).



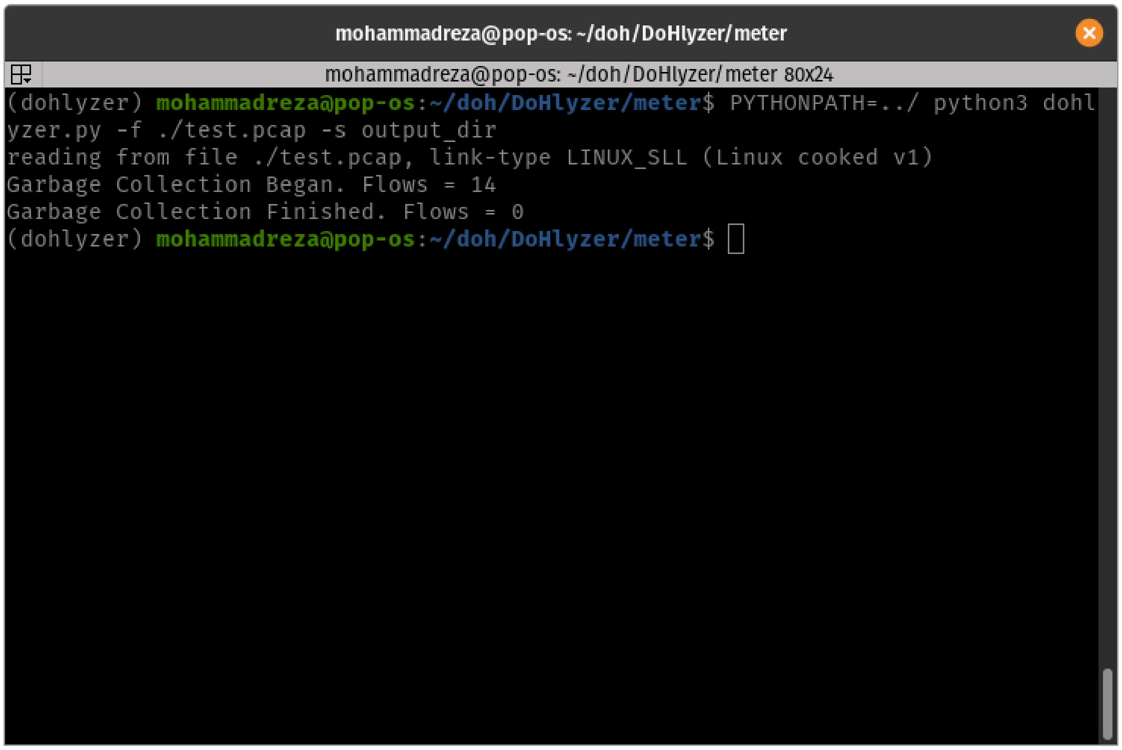
```
SourceIP,DestinationIP,SourcePort,DestinationPort,TimeStamp,Duration,FlowBytesSent,FlowSentRate,F
192.168.20.144,1.1.1.1,34138,443,2020-03-29
19:24:06,460.099462,17092,37.14848942814021,44340,96.37046695785965,52910.41726301343,230.0226451
192.168.20.144,1.1.1.1,34140,443,2020-03-29
19:25:07,460.120022,17367,37.744499629707484,39557,85.9710469195796,45671.38083913708,213.7086353
192.168.20.144,1.1.1.1,34142,443,2020-03-29
19:26:08,461.048141,19393,42.062852607836454,42018,91.1358191551628,50282.341822720264,224.237244
192.168.20.144,1.1.1.1,34144,443,2020-03-29
19:27:08,460.990094,19835,43.026954934957885,44546,96.63114366184189,54874.82528207161,234.253762
192.168.20.144,1.1.1.1,34146,443,2020-03-29
19:28:08,459.419509,19102,41.57855647353452,43807,95.35293809214357,54115.85715139562,232.6281521
192.168.20.144,1.1.1.1,34148,443,2020-03-29
19:29:09,460.08133,16580,36.037106743714205,42721,92.85532190580305,56350.94913036939,237.3835485
192.168.20.144,1.1.1.1,34150,443,2020-03-29
19:30:09,438.903603,15368,35.014522311861725,41748,95.11883637920376,58216.53073300325,241.281020
192.168.20.144,1.1.1.1,34152,443,2020-03-29
19:31:10,377.215293,15145,40.14948566785706,42575,112.86657988174409,62305.78519126374,249.611268
192.168.20.144,1.1.1.1,34154,443,2020-03-29
19:32:11,316.159463,15159,47.94732334170241,39368,124.51944226638568,55147.99055986465,234.836092
192.168.20.144,1.1.1.1,34156,443,2020-03-29
19:33:11,257.430868,14746,57.281397971279816,40943,159.0446410645673,61480.68859649123,247.952996
192.168.20.144,1.1.1.1,34158,443,2020-03-29
19:34:11,197.042784,15738,79.87097868044738,41914,212.7152243240737,49733.532928719,223.010163285
192.168.20.144,1.1.1.1,34160,443,2020-03-29
19:35:12,135.366045,14773,109.1337196118864,38557,284.8350928772426,58309.47237262002 5,241.473543
192.168.20.144,1.1.1.1,34162,443,2020-03-29
19:36:13,74.698817,16670,223.1628380406613,40155,537.5587139485756,58335.407642494865,241.5272399
192.168.20.144,1.1.1.1,34164,443,2020-03-29
19:37:13,15.380172,4514,293.49476715865075,13288,863.9695316801399,123957.20442708333,352.0755663
```

Figure A.3: Statistical features extracted by meter module

A.2.2 Extracting Time-series Features

To extract the time-series features we use `-s` switch and indicate the output directory in our command, as in Figure A.4:

```
# PYTHONPATH=..  python3 dohlyzer.py -f ./test.pcap -s output_dir
```

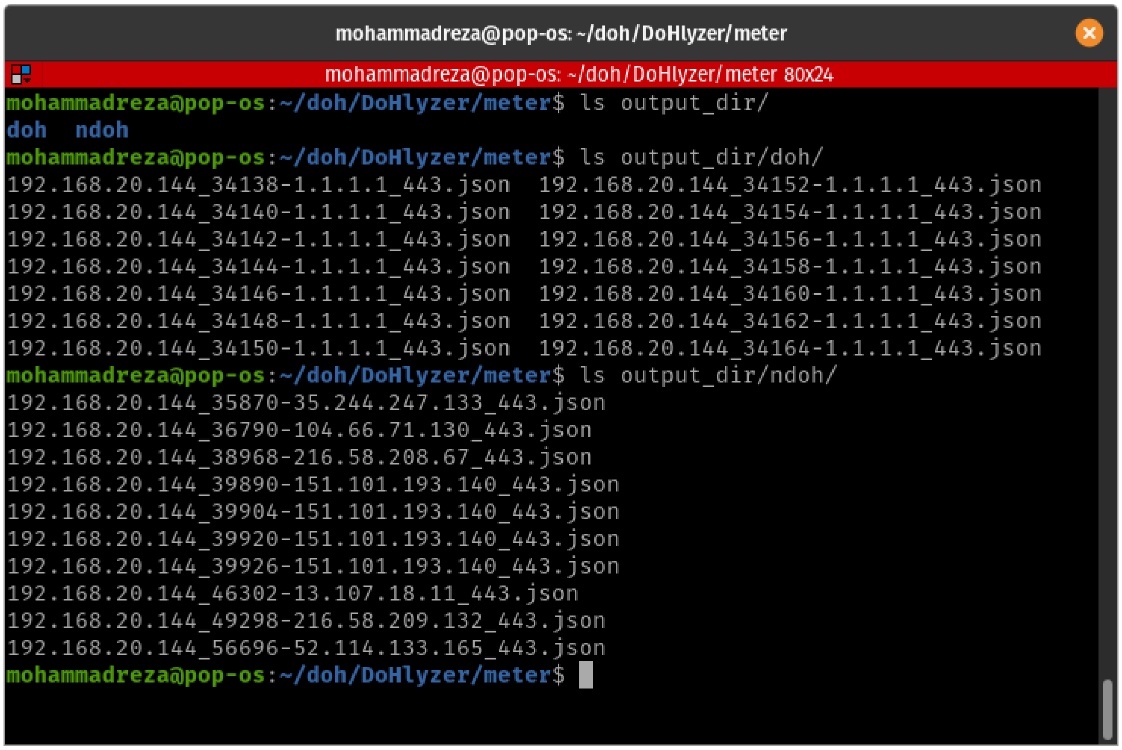


mohammadreza@pop-os: ~/doh/DoHlyzer/meter
mohammadreza@pop-os: ~/doh/DoHlyzer/meter 80x24

```
(dohlyzer) mohammadreza@pop-os:~/doh/DoHlyzer/meter$ PYTHONPATH=../ python3 dohl
yzer.py -f ./test.pcap -s output_dir
reading from file ./test.pcap, link-type LINUX_SLL (Linux cooked v1)
Garbage Collection Began. Flows = 14
Garbage Collection Finished. Flows = 0
(dohlyzer) mohammadreza@pop-os:~/doh/DoHlyzer/meter$ 
```

Figure A.4: Running meter module to extract time-series features

The results are then save in `output_dir` as we requested (See Figure A.5).


mohammadreza@pop-os: ~/doh/DoHlyzer/meter
mohammadreza@pop-os: ~/doh/DoHlyzer/meter 80x24

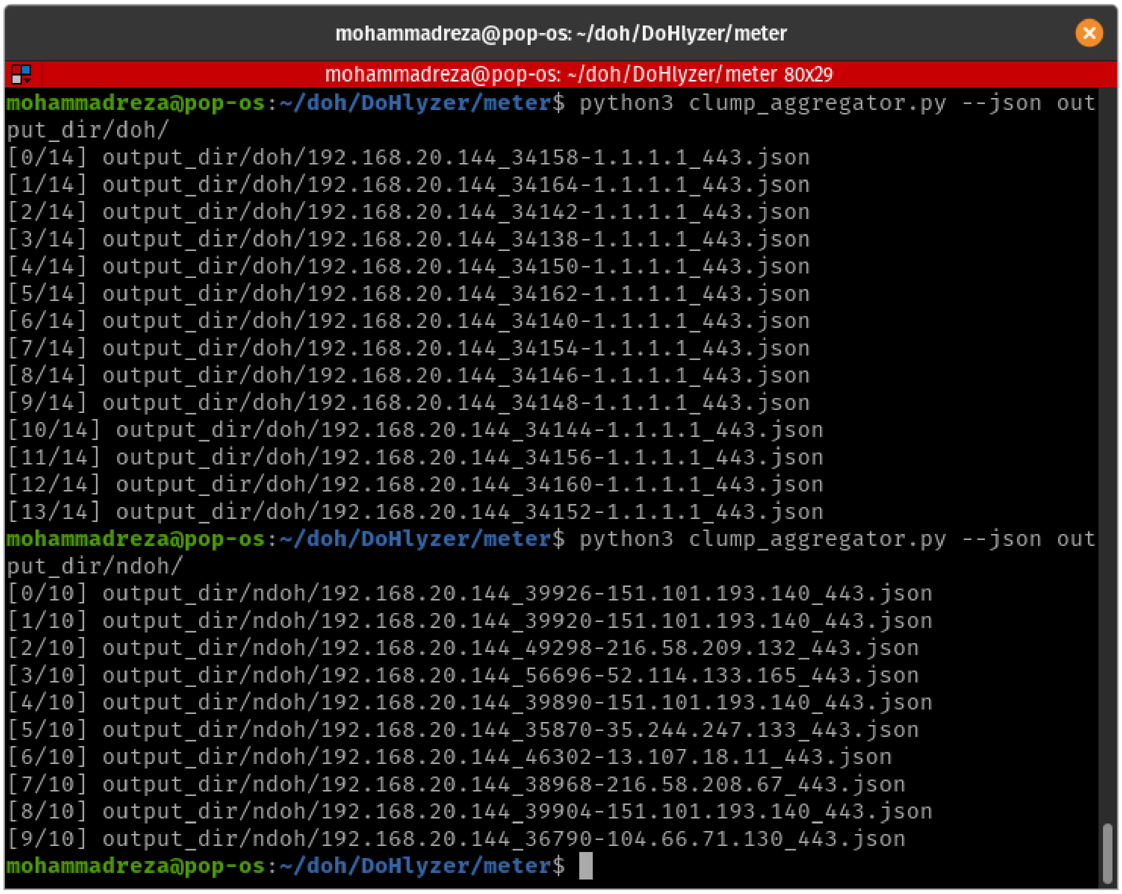
```
mohammadreza@pop-os:~/doh/DoHlyzer/meter$ ls output_dir/
doh  ndoh
mohammadreza@pop-os:~/doh/DoHlyzer/meter$ ls output_dir/doh/
192.168.20.144_34138-1.1.1.1_443.json  192.168.20.144_34152-1.1.1.1_443.json
192.168.20.144_34140-1.1.1.1_443.json  192.168.20.144_34154-1.1.1.1_443.json
192.168.20.144_34142-1.1.1.1_443.json  192.168.20.144_34156-1.1.1.1_443.json
192.168.20.144_34144-1.1.1.1_443.json  192.168.20.144_34158-1.1.1.1_443.json
192.168.20.144_34146-1.1.1.1_443.json  192.168.20.144_34160-1.1.1.1_443.json
192.168.20.144_34148-1.1.1.1_443.json  192.168.20.144_34162-1.1.1.1_443.json
192.168.20.144_34150-1.1.1.1_443.json  192.168.20.144_34164-1.1.1.1_443.json
mohammadreza@pop-os:~/doh/DoHlyzer/meter$ ls output_dir/ndoh/
192.168.20.144_35870-35.244.247.133_443.json
192.168.20.144_36790-104.66.71.130_443.json
192.168.20.144_38968-216.58.208.67_443.json
192.168.20.144_39890-151.101.193.140_443.json
192.168.20.144_39904-151.101.193.140_443.json
192.168.20.144_39920-151.101.193.140_443.json
192.168.20.144_39926-151.101.193.140_443.json
192.168.20.144_46302-13.107.18.11_443.json
192.168.20.144_49298-216.58.209.132_443.json
192.168.20.144_56696-52.114.133.165_443.json
mohammadreza@pop-os:~/doh/DoHlyzer/meter$ 
```

Figure A.5: Time-series features extracted by meter module in `output_dir` directory

As you can see, there is a JSON file for each of the flows of the traffic. To aggregate all of the JSON files in each directory and having one file per label (DoH/non-DoH), we can use the `clump_aggregator.py` script (See Figure A.6):

```
python3 clump_aggregator.py --json output_dir/doh
```

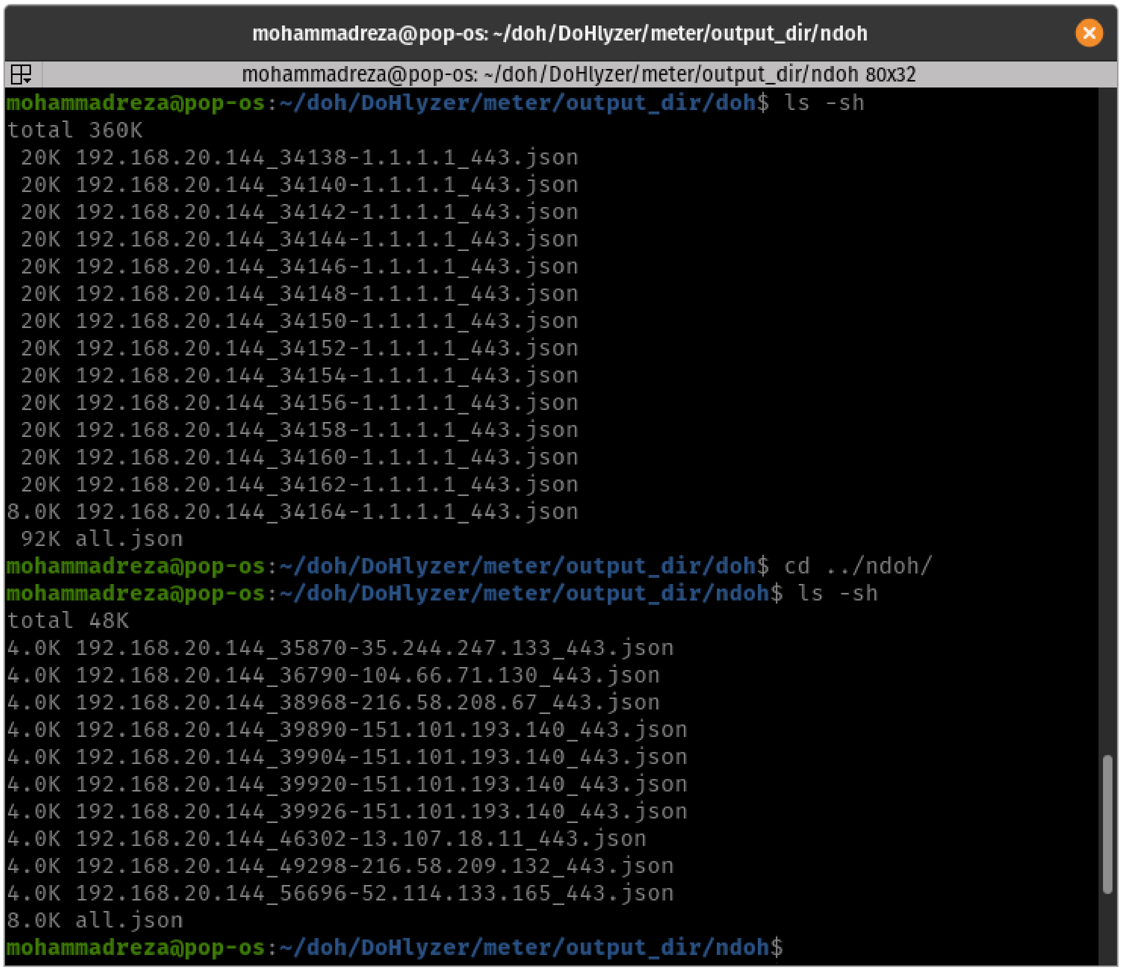
```
python3 clump_aggregator.py --json output_dir/ndoh
```


mohammadreza@pop-os: ~/doh/DoHlyzer/meter
mohammadreza@pop-os: ~/doh/DoHlyzer/meter 80x29

```
mohammadreza@pop-os:~/doh/DoHlyzer/meter$ python3 clump_aggregator.py --json out
put_dir/doh/
[0/14] output_dir/doh/192.168.20.144_34158-1.1.1.1_443.json
[1/14] output_dir/doh/192.168.20.144_34164-1.1.1.1_443.json
[2/14] output_dir/doh/192.168.20.144_34142-1.1.1.1_443.json
[3/14] output_dir/doh/192.168.20.144_34138-1.1.1.1_443.json
[4/14] output_dir/doh/192.168.20.144_34150-1.1.1.1_443.json
[5/14] output_dir/doh/192.168.20.144_34162-1.1.1.1_443.json
[6/14] output_dir/doh/192.168.20.144_34140-1.1.1.1_443.json
[7/14] output_dir/doh/192.168.20.144_34154-1.1.1.1_443.json
[8/14] output_dir/doh/192.168.20.144_34146-1.1.1.1_443.json
[9/14] output_dir/doh/192.168.20.144_34148-1.1.1.1_443.json
[10/14] output_dir/doh/192.168.20.144_34144-1.1.1.1_443.json
[11/14] output_dir/doh/192.168.20.144_34156-1.1.1.1_443.json
[12/14] output_dir/doh/192.168.20.144_34160-1.1.1.1_443.json
[13/14] output_dir/doh/192.168.20.144_34152-1.1.1.1_443.json
mohammadreza@pop-os:~/doh/DoHlyzer/meter$ python3 clump_aggregator.py --json out
put_dir/ndoh/
[0/10] output_dir/ndoh/192.168.20.144_39926-151.101.193.140_443.json
[1/10] output_dir/ndoh/192.168.20.144_39920-151.101.193.140_443.json
[2/10] output_dir/ndoh/192.168.20.144_49298-216.58.209.132_443.json
[3/10] output_dir/ndoh/192.168.20.144_56696-52.114.133.165_443.json
[4/10] output_dir/ndoh/192.168.20.144_39890-151.101.193.140_443.json
[5/10] output_dir/ndoh/192.168.20.144_35870-35.244.247.133_443.json
[6/10] output_dir/ndoh/192.168.20.144_46302-13.107.18.11_443.json
[7/10] output_dir/ndoh/192.168.20.144_38968-216.58.208.67_443.json
[8/10] output_dir/ndoh/192.168.20.144_39904-151.101.193.140_443.json
[9/10] output_dir/ndoh/192.168.20.144_36790-104.66.71.130_443.json
mohammadreza@pop-os:~/doh/DoHlyzer/meter$ 
```

Figure A.6: Using clump aggregator script on the output of previous step

As you can see in Figure A.7, the JSON files are all aggregated into `all.json` in each directory.


mohammadreza@pop-os: ~/doh/DoHlyzer/meter/output_dir/ndoh
mohammadreza@pop-os: ~/doh/DoHlyzer/meter/output_dir/ndoh 80x32

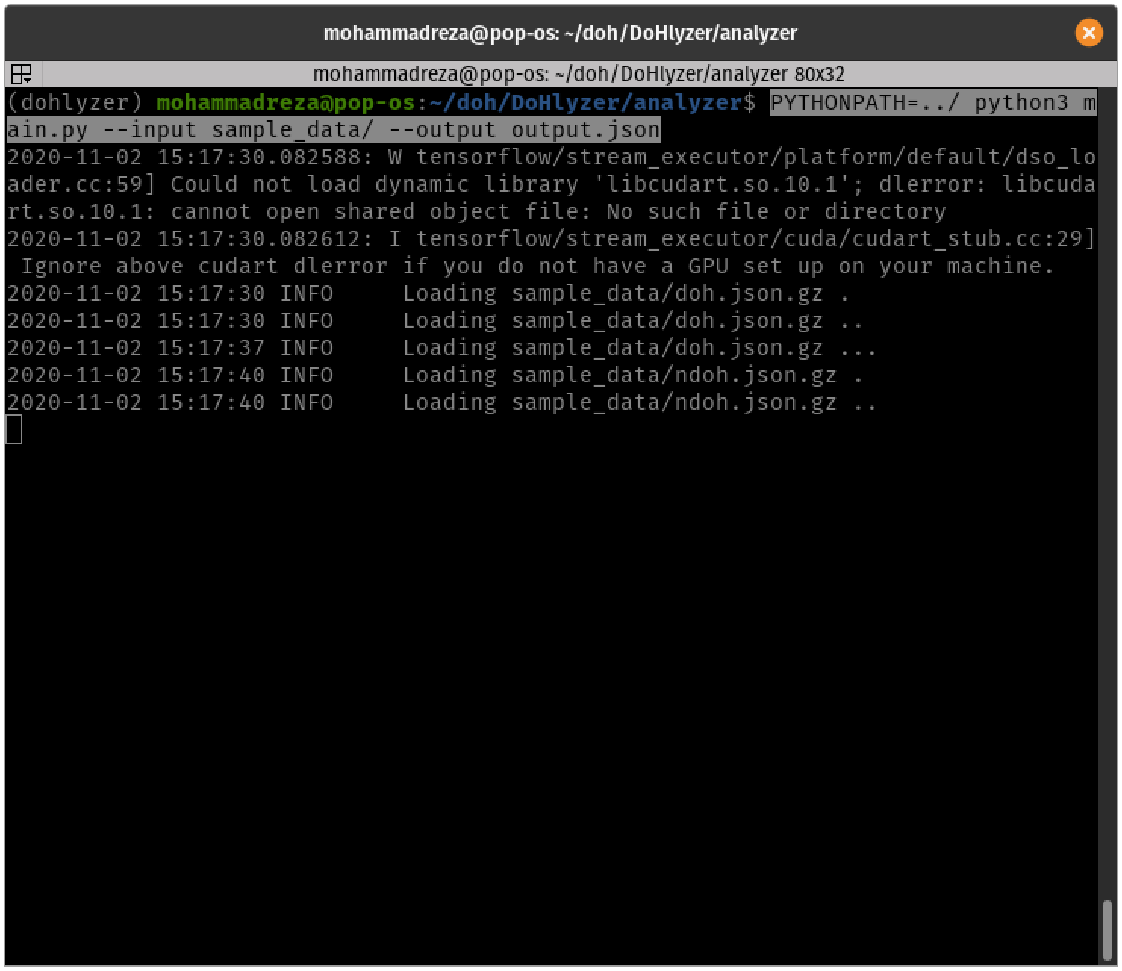
```
mohammadreza@pop-os:~/doh/DoHlyzer/meter/output_dir/doh$ ls -sh
total 360K
 20K 192.168.20.144_34138-1.1.1.1_443.json
 20K 192.168.20.144_34140-1.1.1.1_443.json
 20K 192.168.20.144_34142-1.1.1.1_443.json
 20K 192.168.20.144_34144-1.1.1.1_443.json
 20K 192.168.20.144_34146-1.1.1.1_443.json
 20K 192.168.20.144_34148-1.1.1.1_443.json
 20K 192.168.20.144_34150-1.1.1.1_443.json
 20K 192.168.20.144_34152-1.1.1.1_443.json
 20K 192.168.20.144_34154-1.1.1.1_443.json
 20K 192.168.20.144_34156-1.1.1.1_443.json
 20K 192.168.20.144_34158-1.1.1.1_443.json
 20K 192.168.20.144_34160-1.1.1.1_443.json
 20K 192.168.20.144_34162-1.1.1.1_443.json
8.0K 192.168.20.144_34164-1.1.1.1_443.json
 92K all.json
mohammadreza@pop-os:~/doh/DoHlyzer/meter/output_dir/doh$ cd ../ndoh/
mohammadreza@pop-os:~/doh/DoHlyzer/meter/output_dir/ndoh$ ls -sh
total 48K
4.0K 192.168.20.144_35870-35.244.247.133_443.json
4.0K 192.168.20.144_36790-104.66.71.130_443.json
4.0K 192.168.20.144_38968-216.58.208.67_443.json
4.0K 192.168.20.144_39890-151.101.193.140_443.json
4.0K 192.168.20.144_39904-151.101.193.140_443.json
4.0K 192.168.20.144_39920-151.101.193.140_443.json
4.0K 192.168.20.144_39926-151.101.193.140_443.json
4.0K 192.168.20.144_46302-13.107.18.11_443.json
4.0K 192.168.20.144_49298-216.58.209.132_443.json
4.0K 192.168.20.144_56696-52.114.133.165_443.json
8.0K all.json
mohammadreza@pop-os:~/doh/DoHlyzer/meter/output_dir/ndoh$
```

Figure A.7: Result of clump aggregator script

A.3 DoH Analyzer Module

The analyzer module comes with some sample data to make testing the functionality easier. You could use the following commands to run this module (As seen in Figure A.8): `# cd ../analyzer`

`# PYTHONPATH=.. python3 main.py --input sample_data/ --output output.json`

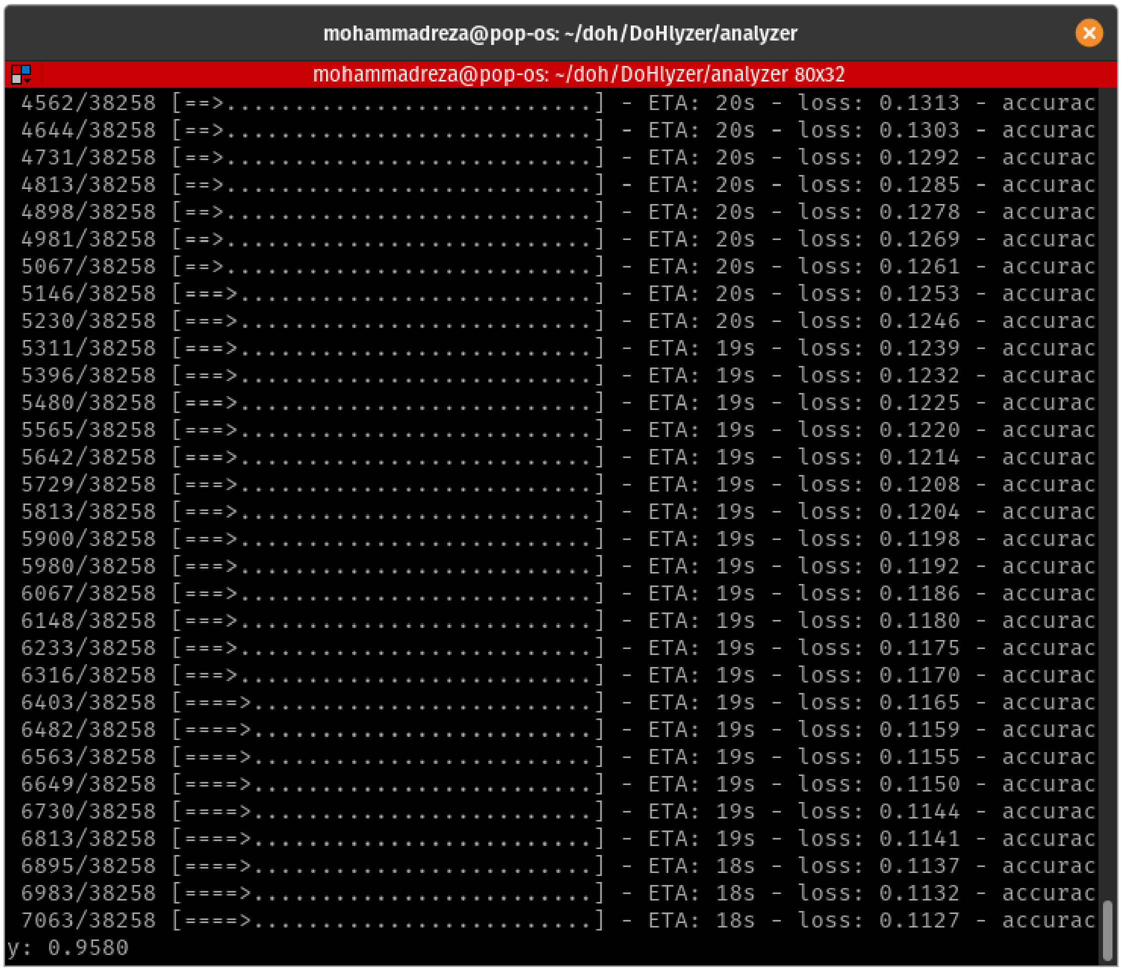


mohammadreza@pop-os: ~/doh/DoHlyzer/analyzer
mohammadreza@pop-os: ~/doh/DoHlyzer/analyzer 80x32

```
(dohlyzer) mohammadreza@pop-os:~/doh/DoHlyzer/analyzer$ PYTHONPATH=../ python3 m
ain.py --input sample_data/ --output output.json
2020-11-02 15:17:30.082588: W tensorflow/stream_executor/platform/default/dso_lo
ader.cc:59] Could not load dynamic library 'libcudart.so.10.1'; dlerror: libcuda
rt.so.10.1: cannot open shared object file: No such file or directory
2020-11-02 15:17:30.082612: I tensorflow/stream_executor/cuda/cudart_stub.cc:29]
 Ignore above cudart dlerror if you do not have a GPU set up on your machine.
2020-11-02 15:17:30 INFO     Loading sample_data/doh.json.gz .
2020-11-02 15:17:30 INFO     Loading sample_data/doh.json.gz ..
2020-11-02 15:17:37 INFO     Loading sample_data/doh.json.gz ...
2020-11-02 15:17:40 INFO     Loading sample_data/ndoh.json.gz .
2020-11-02 15:17:40 INFO     Loading sample_data/ndoh.json.gz ..
```

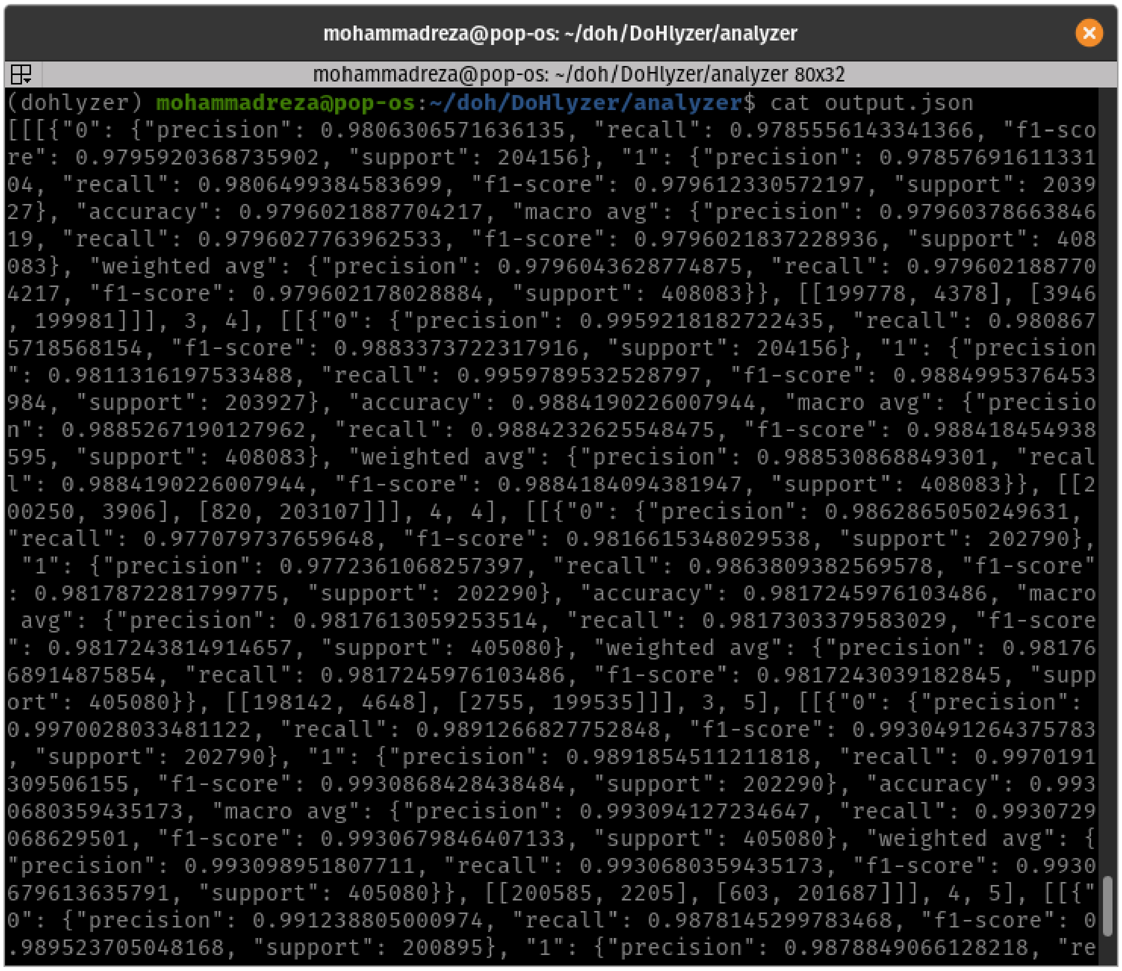
Figure A.8: Using analyzer module to create DNN classifiers

Since this command is training and benchmarking the DNN classifiers, this step may take a while (especially if you don't have GPU support for Tensorflow). You would see something like Figure A.9. The results of the benchmark would be save into a JSON file as seen in Figure A.10.


mohammadreza@pop-os: ~/doh/DoHlyzer/analyzer
mohammadreza@pop-os: ~/doh/DoHlyzer/analyzer 80x32

```
 4562/38258 [==>..........................] - ETA: 20s - loss: 0.1313 - accurac
 4644/38258 [==>..........................] - ETA: 20s - loss: 0.1303 - accurac
 4731/38258 [==>..........................] - ETA: 20s - loss: 0.1292 - accurac
 4813/38258 [==>..........................] - ETA: 20s - loss: 0.1285 - accurac
 4898/38258 [==>..........................] - ETA: 20s - loss: 0.1278 - accurac
 4981/38258 [==>..........................] - ETA: 20s - loss: 0.1269 - accurac
 5067/38258 [==>..........................] - ETA: 20s - loss: 0.1261 - accurac
 5146/38258 [===>.........................] - ETA: 20s - loss: 0.1253 - accurac
 5230/38258 [===>.........................] - ETA: 20s - loss: 0.1246 - accurac
 5311/38258 [===>.........................] - ETA: 19s - loss: 0.1239 - accurac
 5396/38258 [===>.........................] - ETA: 19s - loss: 0.1232 - accurac
 5480/38258 [===>.........................] - ETA: 19s - loss: 0.1225 - accurac
 5565/38258 [===>.........................] - ETA: 19s - loss: 0.1220 - accurac
 5642/38258 [===>.........................] - ETA: 19s - loss: 0.1214 - accurac
 5729/38258 [===>.........................] - ETA: 19s - loss: 0.1208 - accurac
 5813/38258 [===>.........................] - ETA: 19s - loss: 0.1204 - accurac
 5900/38258 [===>.........................] - ETA: 19s - loss: 0.1198 - accurac
 5980/38258 [===>.........................] - ETA: 19s - loss: 0.1192 - accurac
 6067/38258 [===>.........................] - ETA: 19s - loss: 0.1186 - accurac
 6148/38258 [===>.........................] - ETA: 19s - loss: 0.1180 - accurac
 6233/38258 [===>.........................] - ETA: 19s - loss: 0.1175 - accurac
 6316/38258 [===>.........................] - ETA: 19s - loss: 0.1170 - accurac
 6403/38258 [====>........................] - ETA: 19s - loss: 0.1165 - accurac
 6482/38258 [====>........................] - ETA: 19s - loss: 0.1159 - accurac
 6563/38258 [====>........................] - ETA: 19s - loss: 0.1155 - accurac
 6649/38258 [====>........................] - ETA: 19s - loss: 0.1150 - accurac
 6730/38258 [====>........................] - ETA: 19s - loss: 0.1144 - accurac
 6813/38258 [====>........................] - ETA: 19s - loss: 0.1141 - accurac
 6895/38258 [====>........................] - ETA: 18s - loss: 0.1137 - accurac
 6983/38258 [====>........................] - ETA: 18s - loss: 0.1132 - accurac
 7063/38258 [====>........................] - ETA: 18s - loss: 0.1127 - accurac
y: 0.9580
```

Figure A.9: DNN classifiers being trained by Tensorflow



```
mohammadreza@pop-os: ~/doh/DoHlyzer/analyzer
mohammadreza@pop-os: ~/doh/DoHlyzer/analyzer 80x32
(dohlyzer) mohammadreza@pop-os:~/doh/DoHlyzer/analyzer$ cat output.json
[[[{"0": {"precision": 0.9806306571636135, "recall": 0.9785556143341366, "f1-sco
re": 0.9795920368735902, "support": 204156}, "1": {"precision": 0.97857691611331
04, "recall": 0.9806499384583699, "f1-score": 0.979612330572197, "support": 2039
27}, "accuracy": 0.9796021887704217, "macro avg": {"precision": 0.97960378663846
19, "recall": 0.9796027763962533, "f1-score": 0.9796021837228936, "support": 408
083}, "weighted avg": {"precision": 0.979604362877487 5, "recall": 0.979602188770
4217, "f1-score": 0.979602178028884, "support": 408083}}, [[199778, 4378], [3946
, 199981]]], 3, 4], [[{"0": {"precision": 0.9959218182722435, "recall": 0.980867
5718568154, "f1-score": 0.9883373722317916, "support": 204156}, "1": {"precision
": 0.9811316197533488, "recall": 0.9959789532528797, "f1-score": 0.9884995376453
984, "support": 203927}, "accuracy": 0.9884190226007944, "macro avg": {"precisio
n": 0.9885267190127962, "recall": 0.9884232625548475, "f1-score": 0.988418454938
595, "support": 408083}, "weighted avg": {"precision": 0.988530868849301, "recal
l": 0.9884190226007944, "f1-score": 0.9884184094381947, "support": 408083}}, [[2
00250, 3906], [820, 203107]]], 4, 4], [[{"0": {"precision": 0.9862865050249631,
"recall": 0.977079737659648, "f1-score": 0.9816615348029538, "support": 202790},
 "1": {"precision": 0.9772361068257397, "recall": 0.9863809382569578, "f1-score"
: 0.9817872281799775, "support": 202290}, "accuracy": 0.9817245976103486, "macro
 avg": {"precision": 0.9817613059253514, "recall": 0.9817303379583029, "f1-score
": 0.9817243814914657, "support": 405080}, "weighted avg": {"precision": 0.98176
68914875854, "recall": 0.9817245976103486, "f1-score": 0.9817243039182845, "supp
ort": 405080}}, [[198142, 4648], [2755, 199535]]], 3, 5], [[{"0": {"precision":
0.9970028033481122, "recall": 0.9891266827752848, "f1-score": 0.9930491264375783
, "support": 202790}, "1": {"precision": 0.9891854511211818, "recall": 0.9970191
309506155, "f1-score": 0.9930868428438484, "support": 202290}, "accuracy": 0.993
0680359435173, "macro avg": {"precision": 0.993094127234647, "recall": 0.9930729
068629501, "f1-score": 0.9930679846407133, "support": 405080}, "weighted avg": {
"precision": 0.993098951807711, "recall": 0.9930680359435173, "f1-score": 0.9930
679613635791, "support": 405080}}, [[200585, 2205], [603, 201687]]], 4, 5], [[{"
0": {"precision": 0.991238805000974, "recall": 0.9878145299783468, "f1-score": 0
.9895237050481 68, "support": 200895}, "1": {"precision": 0.987884906612821 8, "re
```

Figure A.10: Results of the analyzer module benchmarks

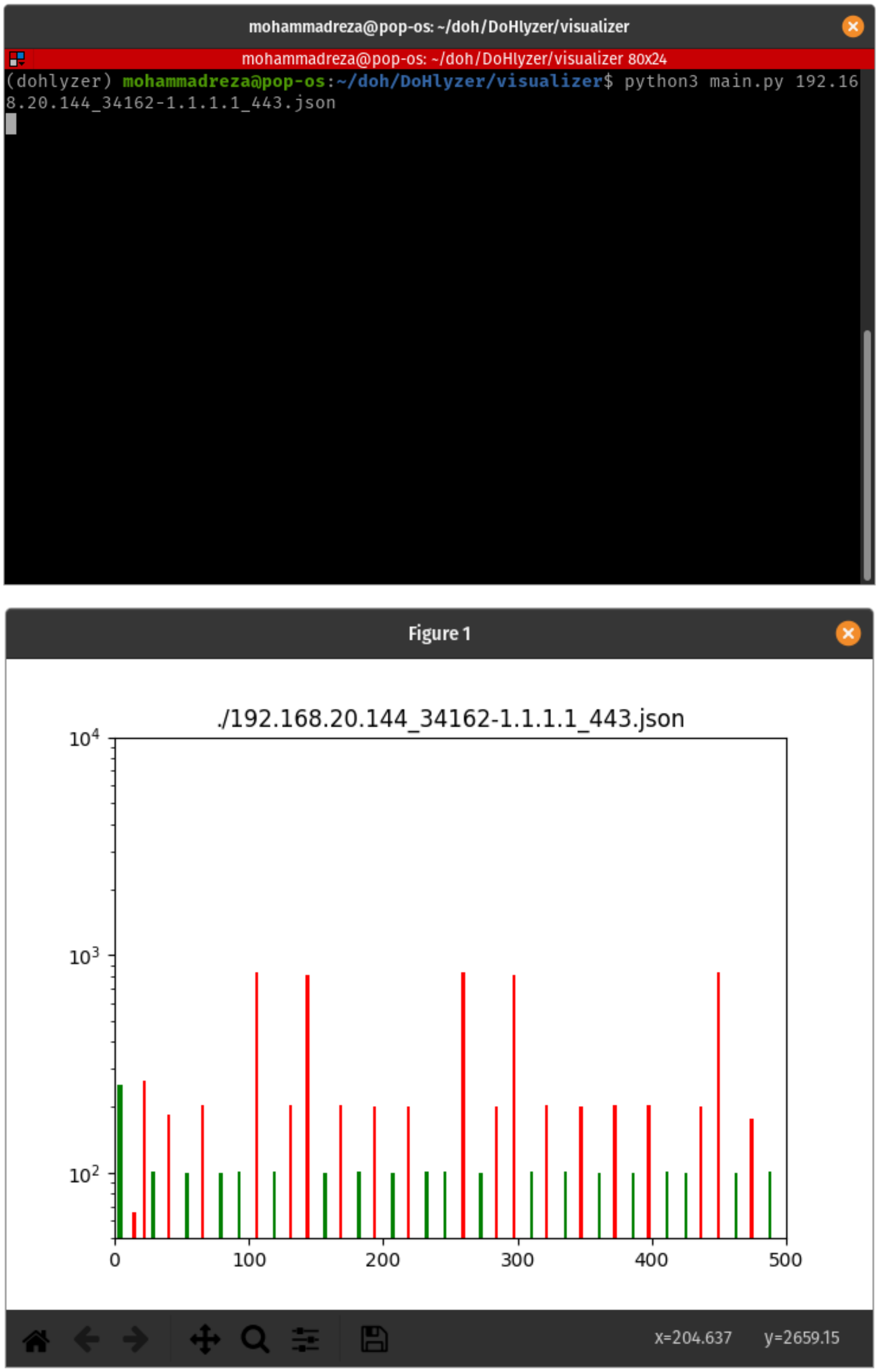
A.4 DoH Visualizer Module

The visualizer module could be used to visualize the JSON files created by meter module from each flow. To visualize a clumps file we should run visualizer module like this:

```
# cd ../visualizer

# python3 main.py input.json
```

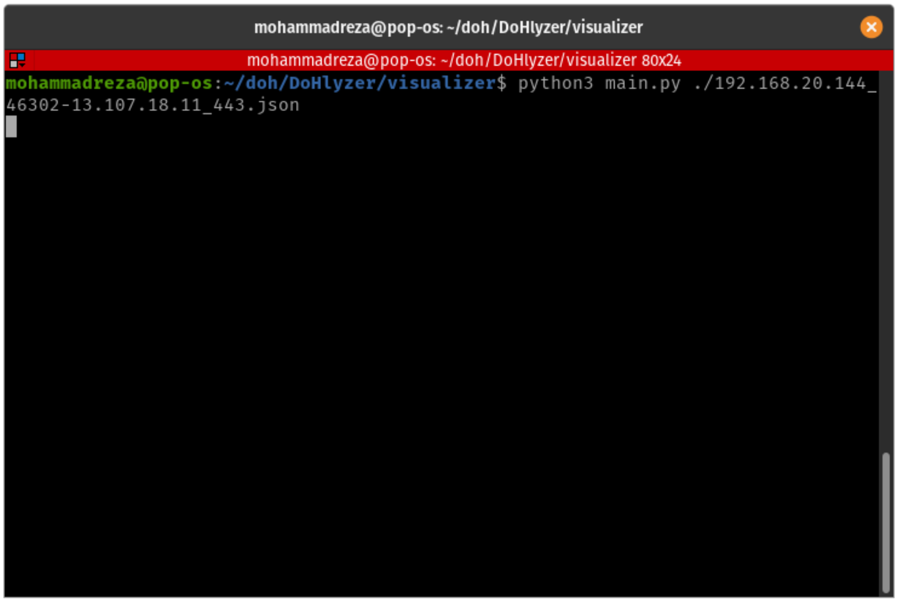
We used results from the meter module to test this module. In Figure A.11 you can see the results of running visualizer module on a clumps file corresponding to a DoH flow. Figure A.12 on the other hands shows the visualization of a non-DoH flow clumps file.



```
(dohlyzer) mohammadreza@pop-os:~/doh/DoHlyzer/visualizer$ python3 main.py 192.16
8.20.144_34162-1.1.1.1_443.json
```

mohammadreza@pop-os: ~/doh/DoHlyzer/visualizer
mohammadreza@pop-os: ~/doh/DoHlyzer/visualizer 80x24
Figure 1
./192.168.20.144_34162-1.1.1.1_443.json
10^4
10^3
10^2
0
100
200
300
400
500
x=204.637
y=2659.15

Figure A.11: Results of running visualization on a DoH flow



```
mohammadreza@pop-os: ~/doh/DoHlyzer/visualizer
mohammadreza@pop-os: ~/doh/DoHlyzer/visualizer 80x24
mohammadreza@pop-os:~/doh/DoHlyzer/visualizer$ python3 main.py ./192.168.20.144_
46302-13.107.18.11_443.json
```

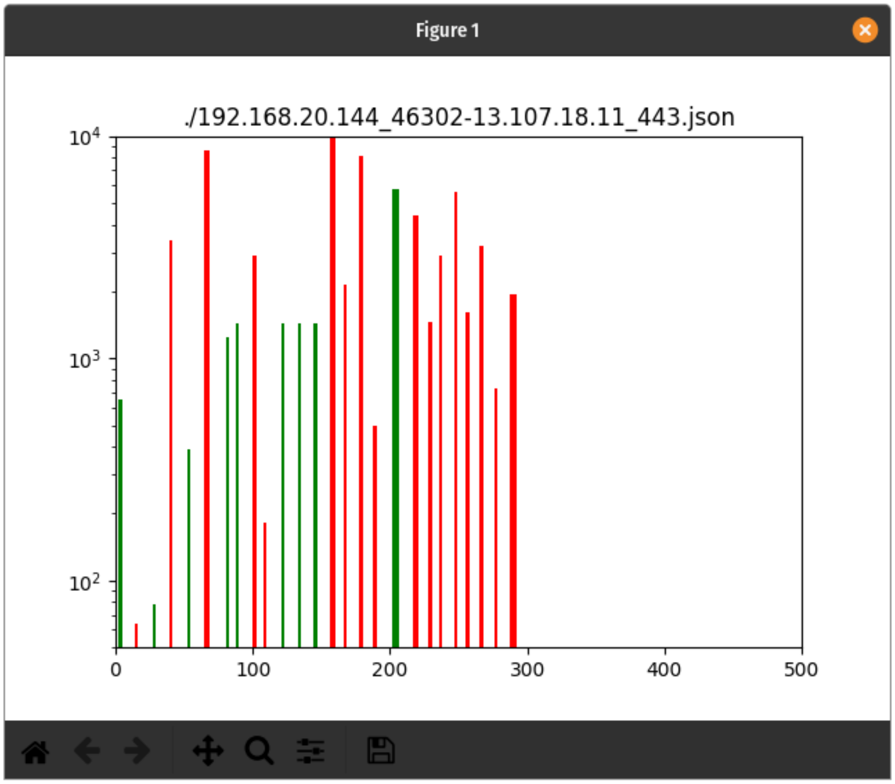


Figure A.12: Results of running visualization on a non-DoH flow

Vita

Candidate's full name: Mohammadreza MontazeriShatoori
University Attended:

- Bachelor of Software Engineering, Sharif University of Technology, IRAN, Tehran, 2012 - 2018

Publications:

- Mohammadreza MontazeriShatoori, Gurdip Kaur, and Arash Habibi Lashkari, "A New Anomaly detection framework for DNS-over-HTTPS (DoH) Tunnel Using Time-series Analysis", Computer and Security, Under review

Conference Presentations:

- Mohammadreza MontazeriShatoori, Logan Davidson, Gurdip Kaur, and Arash Habibi Lashkari, "Detection of DoH Tunnels using Time-series Classification of Encrypted Traffic", The 5th IEEE Cyber Science and Technology Congress, Calgary, Canada, August 2020