Wrapping DNS into HTTP(S): An Empirical Study on Name Resolution in Mobile Applications

Baiyang Li*†‡, Yujia Zhu*†‡✉, Qingyun Liu*†‡, Yong Sun*, Yuedong Zhang§ and Li Guo*†‡
*Institute of Information Engineering, Chinese Academy of Sciences
†National Engineering Research Center of Information Security
‡School of Cyber Security, University of Chinese Academy of Sciences
§National Computer Network Emergency Response Technical Team/Coordination of China
Email: zhuyujia@iie.ac.cn

***Abstract*—Wrapping DNS into HTTP(S) is a promising way to mitigate the privacy and security issues of the traditional DNS. It has been standardized by IETF, i.e., DNS-over-HTTPS (DoH). This approach allows the application to choose open resolvers that it trusts, protecting its activities from potential snooping. Moreover, an application can establish a connection with its resolvers, incorporating specific handles and identifiers for customized use. How is the name resolution process performed on the client side? What are the criteria for an application to choose a resolver? These questions are still unclear.**

In this paper, we examine the application-level name resolution practices of 25 popular apps on Android and iOS platforms, revealing their adoption and usage patterns. We present the following findings: (i) non-standard, self-defined HTTP(S)DNS is more prevalent than DoH in practice, (ii) popular apps tend to use dispersed resolvers, some of which are self-owned, (iii) HTTP(S)DNS usage patterns differ across apps. These findings raise new issues related to the transparency and security of DNS configuration inside apps. We also explore the implications of these changes on the DNS ecosystem and analyze the potential security risks.

***Index Terms*—Name Resolution, Encrypted DNS, Client-Side, Network Measurement**

I. Introduction

The Domain Name System (DNS) is a vital component of the Internet infrastructure that maps user-friendly names to Internet resources. Most Internet activities begin with a DNS query. However, DNS was designed to operate in plaintext, exposing it to various security and privacy threats.

To address this issue, several solutions have been proposed and implemented. One of them is to encapsulate DNS queries and responses in HTTP(S) messages. This approach was initially experimented and named HTTP(S)DNS by developers. In 2018, it was standardized by IETF as a protocol, namely DNS-over-HTTPS (DoH). DoH can be deployed on both systems and applications, and it has shown a tendency to replace the traditional DNS protocol on the client-side.

Prior research on the adoption of DNS has primarily concentrated on DNS servers, with little examination of the client-side. A significant yet neglected aspect is the application's ability to select a preferred resolver that differs from the system default [1]. Indeed, each application has the capability to establish its own specific resolvers and tailor a unique namespace to fulfill its requirements. However, the status of application-level name resolution remains unclear as it has not been examined.

The goal of our study is to examine name resolution patterns on the client-side, which we believe will offer a valuable perspective on DNS utilization. Our research seeks to address the following questions: (i) How do widely-used mobile apps resolve domain names? (ii) Are there any mobile apps that employ self-configuring resolvers? (iii) What are the actual DNS resolution patterns of these apps?

To answer these questions, we conducted an comprehensive analysis of 25 prominent applications and scrutinized their usage patterns in detail. In summary, the contributions of this paper are concluded as follows:

- We provide a summary of the alterations in the conventional name resolution. From the perspective of traffic analysis, we propose a testing methodology to ascertain the DNS resolution methods employed by applications.
- A comprehensive analysis of 25 popular apps was conducted, including both Android and iOS versions. The results indicate that the resolvers configured within the apps tend to be dispersed and that self-use HTTP(S)DNS resolvers are prevalent among these apps.
- We examine the usage patterns of non-standard HTTP(S)DNS adopted by apps and discover that HTTP(S)DNS takes on various forms without a uniform implementation across different apps.
- We contemplate the new issues that these changes will introduce for users, applications, ISPs, and the DNS ecosystem. Potential security implications are discussed and feasible recommendations are provided.

Definitions. *HTTP(S)DNS*: In this study, we differentiate HTTP(S)DNS from DoH based on the RFC criterion. According to RFC 8484 [2], we define the use of HTTP(S) in name resolution without DNS wire format as HTTP(S)DNS. By this criterion, we classify the JSON API provided by Google and Cloudflare's DoH [3], [4] as HTTPSDNS.

Self-use resolvers: In contrast to public DNS resolvers, self-use resolvers refer to private servers specific to an application with their own domain name or IP address. We will focus our attention on self-use resolvers that employ non-standard HTTP(S)DNS.

ISBN 978-3-903176-57-7

Organization. The remainder of this paper is organized as follows. Section II reviews the alterations in the conventional name resolution. Section III and Section IV examine the adoption of HTTP(S)DNS in mobile apps. Section V analyzes usage patterns of apps in practice. Section VI discusses the potential impacts. Section VII summarizes related work, and Section VIII concludes our work.

II. Background

As the demands for performance, security, and privacy continue growing, application-oriented DNS resolution is rising. As shown in Figure 1, DNS resolution is transitioning from local ISP to public DNS and further to application-specific services [5]. In this section, we briefly analyze how changes in protocol affect the name resolution process on the client-side and introduce development trends in name resolution.

A. Conventional Name Resolution

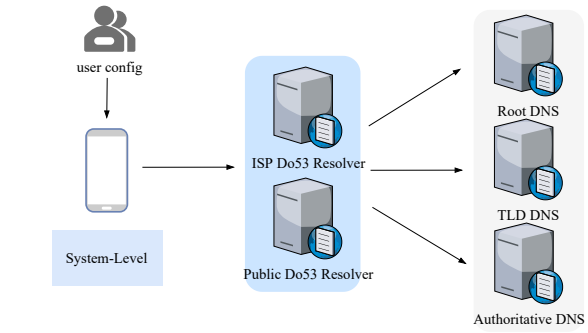
In the conventional name resolution, DNS packets are designed to be transmitted over UDP using port 53 (Do53). Applications are not necessarily cognizant the detail of name resolution. Typically, the application calls the hosting system through the network library API like `gethostbyname`. The hosting system then processes the name resolution task and returns responses to the application [5].

Initially, The name resolution service is typically provided by local ISP resolvers. With the emergence of public DNS resolvers, some users switch their DNS query to public DNS services. At this stage, the application has no decision-making power in the name resolution process other than triggering the initial query [5]. Figure 1(a) illustrates the conventional name resolution scenario in which each app uses the same resolver and will not receive ambiguous answers when sending a query.

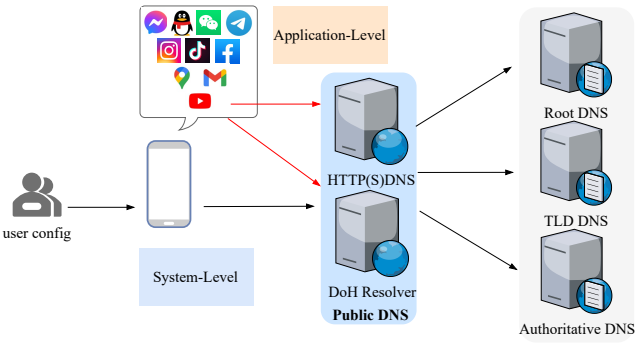
B. New Resolution Approaches

Do53 has faced criticism for its security and privacy deficiencies. Its vulnerability to manipulation can lead to false results. To mitigate these risks, several new extensions have been specified and implemented for improvement. One potential approach is to circumvent the local DNS and prevent DNS injection/hijacking by the resolver. HTTP(S)DNS is a representative solution for countering Do53 hijacking and enabling precise scheduling by replacing the traditional Do53 with the HTTP protocol. To date, HTTP(S)DNS has been promoted and developed by several large Internet companies in China, including Baidu [6], Alibaba [7], and Tencent [8].

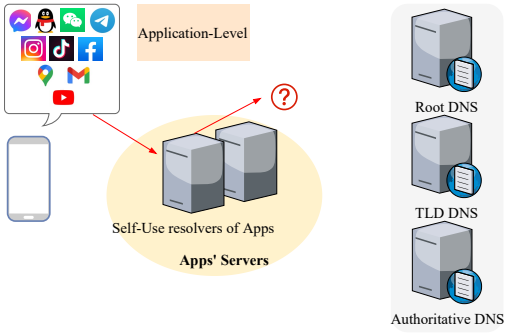
As the demand for DNS privacy rises, encrypted DNS has garnered significant attention. Two new protocols, DNS-over-TLS (DoT) and DNS-over-HTTPS (DoH), have been proposed and standardized by IETF. Of the two protocols, DoH is more appealing to web applications due to its compatibility with the existing network stack. Compared to HTTP(S)DNS, DoH provides a tunnel over HTTPS while preserving the DNS wire-format. Most public DNS resolution providers (e.g., Google, Cloudflare, and Quad9 [9]–[11]) have already established public DoH resolvers.



a. Conventional name resolution



b. Wrapping DNS into HTTP(S) brings changes



c. Application-Specific name resolution

Fig. 1. The development tendency in name resolution on the client-side.

Wrapping DNS into HTTP(S) brings changes to name resolution. As shown in Figure 1(b), the application can conduct DNS transactions in its own manner, which can not necessarily adhere to the system-level DNS configuration. Both public DNS providers and network libraries offer new APIs for developers. Name resolution at the application-level is not always consistent with the system default [1].

C. Application-Specific Name Resolution

If an application is permitted to resolver other than the system default, it is likely to construct its own specific DNS resolvers. In this manner, the application can manage name resolution in an ad hoc fashion, incorporating unique identifiers into DNS messages. It is capable of customizing

a DNS server for self-use to fulfill special requirements [5]. Figure 1(c) presents a scenario in which name resolution is specific to the application. The app initiates a DNS query directly with its self-use resolvers. However, the manner in which these resolvers handle DNS queries remains unknown. They can maintain an exclusive set of DNS resource records that are customized by their respective applications.

III. Methodology

This work aims to reveal application-level name resolution strategies used by mobile apps at the application layer. To this end, we conduct traffic analysis on each app to identify its DNS resolution mechanisms. In this section, we describe on our methodology in detail. Figure 2 illustrates the overview of our methodology, which comprises three steps: data collection, correlation, and detection.

A. Data Collection

Since many apps require user authentication to operate correctly, we need to manually input some information to test their functionality. To emulate the behavior of real users, we interact with the UI of the app to generate DNS requests as much as possible, aiming to cover at least the main features offered by each app. We record the traffic data of each app using the ISP's local resolver as the default Do53 resolver in the experiment.

During the data collection process, we encounter noise interference that affects our analysis. To obtain a clean dataset, we adopt the following countermeasures. On one hand, we collect the background traffic produced by the system first. On the other hand, we only allow one app to access WLAN at a time, to minimize the interference of unrelated processes during testing. After filtering out the system noise, we finally get a relatively clean dataset, which is the basis for our subsequent analysis.

B. Correlation

The challenge in identifying the DNS resolution mechanisms of an app stems from the non-standard implementations of self-use modes. Such a customized mode, developed by app developers, is difficult to identify. To uncover the details, we separate the traffic data of each app into Do53 and the others. Intuitively, if an app only uses Do53 for name resolution, it begins with a Do53 query, receives a response, and then communicates with the target server. We call this case as Do53-Only, which can be formalized as follows:

Given a set of IPs in Do53 RRs $Resolved\_IP = \{rip_1, rip_2, ..., rip_m\}$ and a set of server IPs $Server\_IP = \{sip_1, sip_2, ..., sip_n\}$ extracted from traffic traces of an app. Let T_{rip} stands for the time when the app receives the answer sip from the Do53 response, and T_{sip} represents the time when the app sends the first packet to the target server sip. The traffic traces of the app satisfy the following condition if only Do53 is configured as the protocol:

1) Each server IP should be included in the Do53 RR set, i.e., $Server\_IP \subseteq Resolved\_IP$.
2) The Do53 response should be received before communication with the server. For each server IP sip_a and its related resolved IP rip_b in the RR set (sip_a = rip_b), $T_{rip_b} < T_{sip_a}$.

All tested apps are reinstalled in the measurement to prevent DNS caching from affecting results. If traces of an app do not follow Do53-only, protocols other than Do53 can be involved in the name resolution process. But this is not the only case of deviation. The peer IP addresses can be obtained in another way besides DNS. Possible situations that we consider against Do53-only can be summarized as follows:

- *Not using Domain Names.* The app establishes server connection via IP directly without using domain names. It can contain several seed IP addresses for a start. For instance, service with statistic IP addresses, or acquire IP addresses by other services, like P2P.
- *New Resolution Methods.* The app still uses domain names and requires the name resolution process. It configures other protocols to replace Do53. This case is what we focus on in this paper.

As the above scenarios are common in applications, the results from correlation turn out to be inadequate. Additional detection is needed to specify further.

C. Detection

The correlation procedure classified apps into two groups, Do53-only or not. The detection procedure aims to further validate the rest of the apps in the not Do53-only group and find potential servers used for name resolution. In general, three steps are involved:

1) Open Resolvers Filtering: We use a resolver list as a filter to discover potential resolvers in an app's traffic. Therefore, an Internet-wide scanning is needed to locate open resolvers. We utilize the probing results of port 443 across all IPv4 addresses provided by Rapid7 Labs [12], scanning for servers with TCP/443 open. Common DoH path templates (/dns-query, /resolve, and /doh, etc) are considered.

Unlike DoH, active scanning is limited in detecting HTTP(S)DNS resolvers due to the lack of a unified standard. HTTP(S)DNS is more like a web service than DNS, which significantly increases the difficulty of detection. We search and register several public HTTP(S)DNS services and then collect IP addresses of resolvers. Finally, we established a set of IPs containing open DoH and HTTP(S)DNS to find resolvers in traffic traces.

2) String Matching: We use the string matching method to find unknown resolvers served for an app. Keywords to be matched consist of two parts:

- IP addresses do not belong to the Do53 RR set, i.e., $sip \notin Resolved\_IP$.
- Common words that appear in URI templates (e.g., doh, dns, httpdns, dn, domain and resolve, etc.).

In order to decrypt the contents of encrypted traffic, we deployed a HTTPSProxy and relaunched the rest apps to inspect the contents of the packets and check if they match.

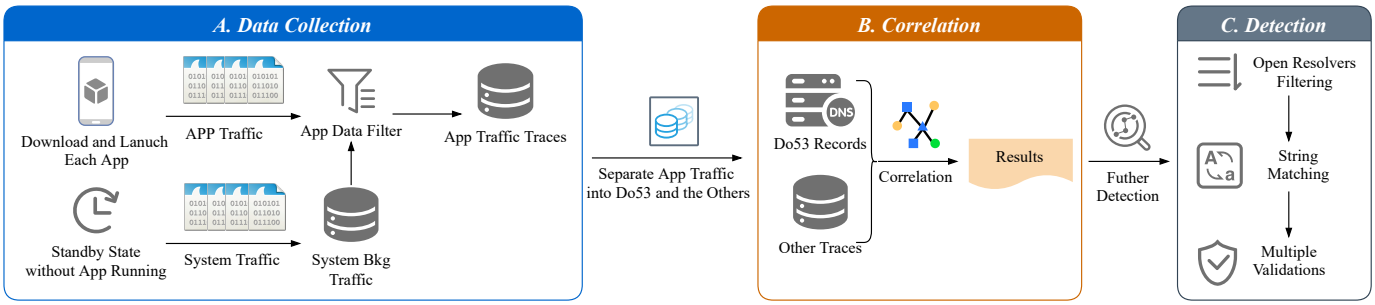


Fig. 2. Methodological overview.

Common transformations are considered during matching, e.g., upper/lower case, Base64 encoding. This step detects private DNS resolvers and reveals the detail of the protocol format.

3) Multiple Validations: Apps can configure SSL pinning to prevent HTTPSProxy from decrypting the traffic. Moreover, Android Nougat has changed the way of handling trusted certificate authorities (CAs). By default, any user-add CAs are no longer trusted by the app [13]. We take some measures to further verify our inference in this step:

- We recheck domain names that match the keywords in the previous step if decryption fails. We send DoH queries to servers with these domains. If a DoH response is received, we consider the server as a resolver.
- The encryption policies of some apps are inconsistent across platforms. By comparing app traffic data collected on different platforms (mainly Android and iOS), we try to infer uncertain versions from the confirmed ones.
- Some apps disclose technical details in their blog post. By searching this extra information, we can determine the result with the help of developers.

To sum up, our method to identify DNS resolvers used by an app consists of four steps. Figure 3 depicts a flow chart of the test for each app. In the correlation step (S_1), apps that do not follow Do53-Only will be selected. The open resolver filtering step (S_2) and the string matching step (S_3) finds public and self-use resolvers. Note that S_2 and S_3 are in parallel since public and self-use resolvers can be both configured in some apps. The final step, Multiple validations (S_4), complements the previous steps. After the four steps, the remaining applications that could not be accurately identified will be categorized as ***Others***.

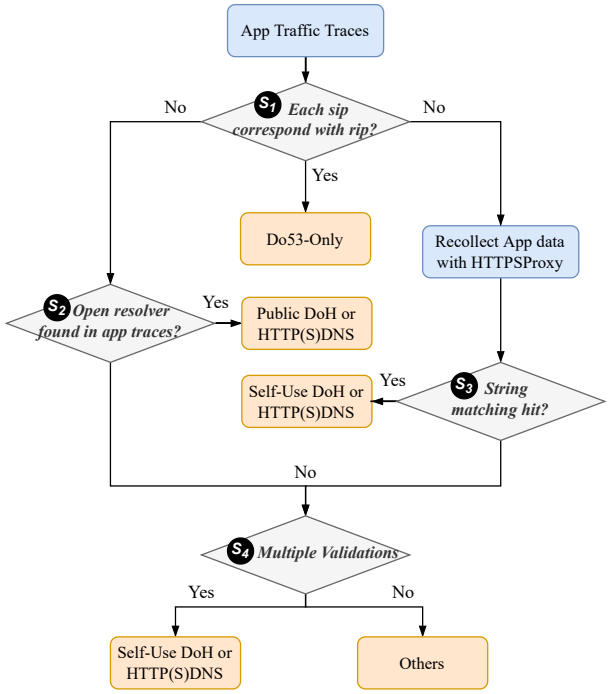


Fig. 3. A Flow Chart of the Test for Each App.

IV. Results

In this section, we first provide an overview of the apps that we measured in our study. Next, we present our findings on the adoption of HTTP(S)DNS by mobile apps from the client-side perspective.

A. Test Apps

To analyze the current state of mobile applications, we selected 25 representative ones from various usage categories, based on their high installs and wide user coverage. We believe that the top-ranked apps are more likely to adopt new protocols and influence other apps in their category. The features of these 25 apps are shown in Table I, along with their ranking in the Apple App Store.

We conducted the measurement in an IPv4 network environment and used the latest versions of Android and iOS applications, covering the listed features. Each measurement were repeated several times to ensure the validity of the results.

B. Prevalence of Adoption

1) Apps Using HTTP(S)DNS: Table I shows an overview of our measurement for 25 apps. We find that HTTP(S)DNS is widely used in practice. Only seven apps follow the Do53-only pattern. About 14 apps, or roughly 60% of our test set, use HTTP(S)DNS in the name resolution process. We also observe regional differences in HTTP(S)DNS usage. Almost all the apps we tested in China ranking use it. This can be due to the early support for HTTP(S)DNS by some public DNS

TABLE I
TEST APPS AND OVERALL ANALYSIS RESULTS.

category	App Name	Features	# of Ranks	Result of Each Step[1] S_1	S_2	S_3	S_4	Overall Results[2]
Chat	Messenger	send message, make audio video calls and view websites	US (6)	✗|✗	✗|✗	✗|✗	✗|✗	④|④
	Telegram		US (41)	✗|✗	✓|✓	✗|✗	-|-	②|②
	WeChat		CN (3)	✗|✗	✗|✗	✗|✗	✓|✓	③|③
	QQ		CN (8)	✗|✗	✗|✗	✗|✗	✗|✗	④|④
Social	Instagram	get updates, send posts and view websites	US (3)	✗|✗	✗|✗	✗|✗	✗|✗	④|④
	Facebook		US (5)	✗|✗	✗|✗	✗|✗	✗|✗	④|④
	Twitter		US (20)	✓|✓	-|-	-|-	-|-	①|①
	weibo		CN (37)	✗|✗	✗|✗	✗|✗	✓|✓	③|③
Shopping	Amazon	Browse products comments, add goods to chart and submit orders	US (7)	✓|✓	-|-	-|-	-|-	①|①
	Taobao		CN (5)	✗|✗	✗|✗	✓|✓	-|-	③|③
	Meituan		CN (14)	✗|✗	✗|✗	✓|✓	-|-	③|③
	JD		CN (15)	✗|✗	✗|✗	✓|✓	-|-	③|③
Video	YouTube	search watch videos, view comments	US (2)	✓|✓	-|-	-|-	-|-	①|①
	Douyin[3]		CN (6)	✗|✗	✓|✗	✗|✗	✓|✓	②③|③
	Tencent Video		CN (20)	✗|✗	✗|✗	✓|✓	-|-	③|③
	iQiyi		CN (28)	✗|✗	✗|✗	✓|✓	-|-	③|③
	BiliBili		CN (29)	✗|✗	✓|✓	✓|✓	-|-	②|②
Payment	PayPal	Add cards, transfer and payment	US (32)	✓|✓	-|-	-|-	-|-	①|①
	AliPay		CN (4)	✗|✗	✗|✗	✓|✓	-|-	③|③
Map	Google Map	Navigate places	US (9)	✓|✓	✗|✗	-|-	-|-	①|①
	AMap		CN (10)	✗|✗	✗|✗	✓|✓	-|-	③|③
Email	GMail	send receive mails, view websites	US (11)	✓|✓	✗|✗	-|-	-|-	①|①
	QQ Mail		CN (42)	✗|✗	✗|✗	✓|✗	-|✗	③|④
News	Reddit	Search for hot posts, leave comments	US (43)	✓|✓	-|-	-|-	-|-	①|①
	Zhihu		CN (44)	✗|✗	✓|✗	✓|✓	-|-	②③|③

[1] Symbols: ✓ means Yes; ✗ means No; '-' means test has end in previous step.
Cell format: <Android|iOS> means that the left one is the result on Android, and the right one is the result on iOS.
[2] Serial numbers: ① stands for Do53-Only; ② stands for public HTTP(S)DNS adoption in the app; ③ stands for self-use HTTP(S)DNS adoption; ②③ stands for both public and self-use HTTP(S)DNS adoption; ④ stands for others.
[3] Also known as Tiktok of China.

providers in China and the availability of SDKs. In contrast, in the US ranking, only *Telegram* uses HTTP(S)DNS, while the other apps do not.

Five apps fall into the ***Others*** category. Among them, *QQ* and the iOS version of *QQ Mail* do not pass any of the four test steps. The other three apps, *Facebook*, *Messenger* and *Instagram* are difficult to classify due to decryption failures. Our results represent the lower bound of apps that use HTTP(S)DNS.

2) HTTP(S)DNS Servers: Our findings reveal that some apps have developed their own servers for name resolution. As explained in Section III, the resolver filtering step (S_2) identifies open resolvers. The string matching step (S_3) and the multiple validations step (S_4) can detect private, self-use ones. Comparing the results in each step, we notice that S_3 finds more apps, which indicates that self-use resolvers are developed and applied.

As shown in table I, among 14 apps using HTTP(S)DNS, only *BiliBili* and *Telegram*, use public DNS services. *BiliBili* adopts AliCloud and Tencent HTTPDNS. *Telegram* sends queries via JSON API provided by Google and Cloudflare, which can be substantiated in its source code.

Twelve apps deploy resolvers for self-use, accounting for a large proportion (above 85%) of adoption. One possible reason for using self-use resolvers is that developers can customize both content and delivery in the name resolution process to suit their specific needs. The results of measurement indicate that self-use resolver is a new trend.

3) iOS vs. Android: Most apps have the same behavior on iOS and Android platforms, except for *QQ Mail*, which uses HTTPDNS on Android but not on iOS. We observe other minor differences between Android and iOS versions in *BiliBili*, *Douyin*, and *Zhihu*. *BiliBili* uses AliCloud HTTPDNS, and also uses Tencent HTTPDNS on its Android version. For *Douyin* and *Zhihu*, AliCloud HTTPSDNS is used on their Android versions but not detected on iOS. We speculate that that the relevant service API is available in Android versions but not in iOS.

V. USAGE PATTERNS ANALYSIS

In this section, we examine the apps that use HTTP(S)DNS, analyzing their usage patterns from the view of specification and implementation. Unlike the standardized DoH guidance

proposed by the IETF, various patterns of HTTP(S)DNS exist in different apps.

A. Bootstrapping

We test how an app initializes an HTTP(S)DNS request. It depends on the service API provided by the resolver. Generally, the client is configured with a URI template. As Figure 4 illustrates, an IP-based URI `(w.x.y.z/resolve?)` is directly accessed by clients without name resolution, which can be hard-coded in apps. Otherwise, a URI containing the hostname `(dns.app.com/resolve?)` needs to be resolved first, i.e., in the bootstrapping step. This step is usually done by the default Do53 resolver of the device, bringing a potential risk of a downgrade attack [14].

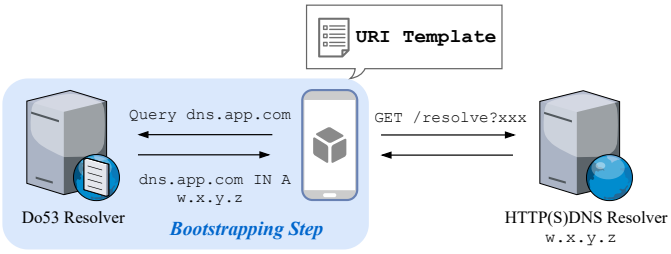


Fig. 4. Bootstrapping step.

TABLE II
URI TEMPLATES OF HTTP(S)DNS

App name	URI Template
Telegram	https://dns.google.com/resolve https://mozilla.cloudflare-dns.com/dns-query
WeChat	http://dns.weixin.qq.com/mmtls
Weibo	http://39.97.130.51/encry_params
Taobao	http://amdc.m.taobao.com/amdc/mobiledispatch
Meituan	http://103.37.142.166/fetch https://httpdns.meituan.com/fetch
JD	https://dns.jd.com/v6/d https://101.124.19.122/v6/d
Douyin	https://dig.bdurl.net
Tencent Video	http://182.254.116.116/d
iQiyi	http://doh.iqiyi.com/md
BiliBili	http://203.197.1.66/resolve http://119.29.29.29/d
Alipay	http://amdc.alipay.com/squery
AMap	http://amdc.m.taobao.com/amdc/mobiledispatch
QQ Mail	http://182.254.116.117/d
Zhihu	https://118.89.204.198/resolve

We examine the apps by combining the results in S_1, S_2, and S_3 of the test flow. Table II summarizes URI templates used in apps. Nine out of 14 apps use hostnames, so they need to send additional Do53 requests before establishing connections with their resolvers. Seven apps access their resolver directly without an additional hostname resolution. Two types of URI are both configured in *JD* and *Meituan*.

B. Encryption Preferences

Not all apps that use HTTP(S)DNS implement encryption. Among the 14 apps, about half of them encrypt DNS messages. We discover two kinds of encryption schemes of HTTPDNS in our measurement. One is by implementing TLS on HTTPDNS, i.e., HTTPSDNS, which secures the entire HTTP Layer. *Douyin*, *Zhihu*, *JD*, *Telegram*, and *Meituan*'s hostname-URI are in HTTPSDNS mode.

The other one is by encrypting partial HTTP elements. These apps encrypt some personal parameters in the HTTP header and body. For example, the Android version of *Bilibili* configures Tencent HTTPDNS, which supports DES or AES encryption for HTTP query parameters and responses. *Weibo*'s HTTPDNS uses this method as well. *Telegram* re-encrypts TXT records of its domain. A private protocol, MMTLS [15] is adopted by *WeChat* to encrypt the HTTP message body. These encrypted contents cannot be accessed through HTTPSProxy.

However, the rest of the 14 apps are in plaintext. They only change Do53 to HTTP. Due to the lack of encryption mechanisms, security and privacy risks still exist. Moreover, these default configurations of apps can be conflicted with the requirements of privacy-sensitive users. We will discuss it further in Section VI.

C. Protocol Formats

HTTP(S)DNS is quite flexible, and developers can customize HTTP(S)DNS to suit special needs. We examine the protocol formats of implementations.

HTTP(S)DNS Requests. Figure 5(a) shows examples of batch queries in packets. Both HTTP GET and POST methods are founded in HTTP(S)DNS requests. Public HTTP(S)DNS providers like AliCloud, Tencent, and Baidu mainly support the HTTP GET method, while private HTTPDNS servers built by apps have no fixed choice. This can be because the GET method is more cache-friendly for HTTP, while the POST method can meet the need to pass specific parameters generated by the app.

In addition, query parameters are different among apps without a standard format. Developers fully leverage the feature set of HTTP protocol. Headers of HTTP(S)DNS requests contain a large amount of information, especially self-use resolvers. The User-Agent, Cookie, and Accept-Language request header fields convey specific information about the client version, platform, or network carrier. As we stated in Section V-B, encryption is inadequate in part HTTP(S)DNS implementation. Although these resolvers are applied to their respective apps, the risk of leaking user privacy has increased.

HTTP(S)DNS Responses. Responses from HTTP(S)DNS vary a lot in apps as well. A common, widely accepted format is serialized in JSON. However, responses in JSON format contain different key-value pairs. Figure 5 (b) shows examples. Google provides a relatively comprehensive DNS response

```
① GET /ID/d?host=app.com&query=4,6
② GET /d?dn=app.cpm
③ GET /fetch?dm=app.com&type=ipv4
④ GET /resolve?name=app.com
⑤ POST /md?business=doh_android&s=1
  {"ttl":1,"query":[{"dn":"app.com","qtype":"a"}]}
⑥ GET /ID/resolve?host=app1.com,app2.com
⑦ GET /d?dn=app1.com,app2.com
⑧ POST /amdc/mobiledispatch?{app parameters}
   domain=app1.com app2.com app3.com...
⑨ GET /v6/b?{app parameters}
⑩ POST /squery
  {app parameters}
```

a. HTTP(S)DNS Requests

```
① {
    "Status": 0,
    "TC": false,
    "RD": true,
    "RA": true,
    "AD": false,
    "CD": false,
    "Question": [{
      "name": "app.com.",
      "type": 1
    }],
    "Answer": [{
      "name": "app.com.",
      "type": 1,
      "TTL": 60,
      "data": "w.x.y.z"
    }]
  }
```

```
② {
    "dns": {
      "host": "app.com",
      "client_ip":"a.b.c.d",
      "ips": ["w.x.y.z"],
      "type": 1,
      "ttl": 36,
      "origin_ttl": 60
    }
  }

③ w.x.y.z,60
```

b. HTTP(S)DNS Responses

Fig. 5. Examples of HTTP(S)DNS message.

example in JSON (item ①). Compared to item ①, a fair part of apps receive a partial resource record (RR). For example, a response only contains IPv4 addresses, missing type and TTL fields of the RR (item ③). The response is only applicable to the specific application.

The Number of Queried Domains per Packet. A DNS request or response packet usually contains one domain name to be queried in Do53, and this convention also applies to DoH. But it changes in HTTP(S)DNS. Multiple domain names can be included in an HTTP(S)DNS request packet, depending on the implementation of the resolver.

As shown in Figure 5 (a), request items ① to ⑤ contain one query, and items ⑥ to ⑧ include two or more domain names. As most HTTP(S)DNS servers keep using HTTP/1.1, we speculate that service providers improve performance by involving multiple domains to achieve a reuse effect.

Items ⑨ and ⑩ represent another pattern in apps in which domain names are omitted in the request. Based on request parameters issued by the client, the resolver pre-provisions a mapping list that contains several DNS RRs. This approach is similar to the feature of HTTP/2 server push, with the difference being that the client still needs to initiate the request. In a way, it reflects a trend of DNS from pull to push.

D. Resolved Domain Names

1) Proportion: More than half of the apps in our measurement use both Do53 and HTTP(S)DNS. We estimate the contribution of HTTP(S)DNS to figure out the role it plays in the name resolution process.

Figure 6 compares the approximate number of domain names resolved by HTTP(S)DNS with Do53. Our statistics cover ten apps, of which the details of HTTP(S)DNS are available in the iOS version (14 apps in total, excluding *Douyin*, *Weibo*, and *WeChat* undecipherable, *QQ Mail* not found in iOS). Since the DNS requests of external domains are generally sent by Do53, domain names corresponding to the web pages that we visit in some apps (Chat, Social, and Email categories) are not considered.

As we can see, HTTP(S)DNS accounts for a relatively small percentage in overall name resolution, and most of the domain names are still resolved by Do53. Apps seem to follow the rules saying that do not put all DNS in one basket. The proportion of HTTP(S)DNS varies. *Telegram* is a special one that rarely sends DNS queries, and most of them are triggered in bootstrapping step. Except for *Alipay*, other apps send more DNS requests through Do53.

Although there are some interference factors: some domains outside of the listed features will not be triggered, and proxies may affect the behavior of the application. Our statistics reveal some trends. Do53 is still the main choice for apps, and HTTP(S)DNS has played a role in the name resolution process.

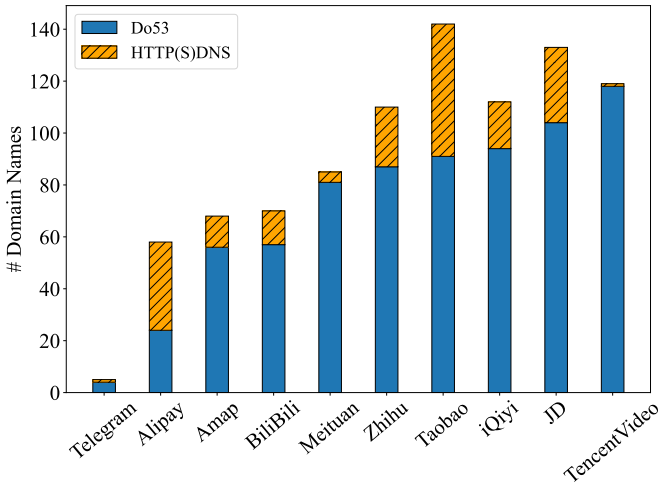


Fig. 6. Number of domain names resolved by Do53 and HTTPDNS.

2) Consistency Comparison: We compare the responses of Do53 and HTTP(S)DNS to discover the differences in the resolution results. For each domain name resolved by HTTP(S)DNS, the comparison is set up as follows:

1) A Do53 lookup is performed first to check if it is resolvable.
2) The prefixes of the IP addresses in HTTP(S)DNS responses are organized by utilizing the RouteViews Prefix to AS mappings Dataset [16].

3) We compare the Do53 response of the domain name with the prefixes in step 2) to see if an IPv4 address (A Type) or an IPv6 address (AAAA Type) is in the subnet set.

Three possible results of the comparison include:

- ***Totally Consistent (All)***: All IPs in the Do53 response are in the subnet set of the HTTP(S)DNS response.
- ***Partially Consistent (Part)***: Some IPs in the Do53 response are in the subnet set, while the others are not.
- ***Completely inconsistent (None)***: None of the Do53 response IPs is in the subnet set.

TABLE III
RESULTS OF THE CONSISTENCY COMPARISON

Query Type	# Domain Names				
	Count	All	Part	None	Missing
A	185	94	13	76	2
AAAA	7	2	0	1	4

In our measurement, 186 domain names are resolved by HTTP(S)DNS. HTTP(S)DNS responses contain A type records of 185 domain names, AAAA type records of seven domain names, and TXT type records of one domain. The TXT type requests are sent by *Telegram*, whose DNS records are encrypted. We are more concerned with A and AAAA records.

As shown in Table III, from the perspective of A type records, about half of domain names' records resolved by Do53 and HTTP(S)DNS are totally consistent. Meanwhile, more than a third of domain names have completely inconsistent records. Since self-use resolvers have direct access to the client IP, they can return more accurate scheduling results to improve user performance, making inconsistent results. Seven domains query AAAA records through HTTP(S)DNS, and the number is relatively small compared to A records.

We also notice domains missing A or AAAA records in Do53. We suspect that these domain names are private, or they miss configurations in DNS RR. We will discuss it in Section VI.

VI. DISCUSSION

In this section, We discuss the implications of configuring self-use resolvers in mobile apps. We consider several key actors involved in the name resolution process.

A. Users

Configurations Conflict between User and Apps. In most cases, the way one app resolves domain names is non-transparent to users. Users do not know which resolvers are configured within the app since it does not offer the option. This creates conflict. The user wrongly assumes that the preferred resolvers are in effect, but some apps do not follow the system configuration. Apps will leak DNS records to pre-set resolvers. Moreover, if an app deploys HTTP(S)DNS without encryption, the implementation can not satisfy the privacy needs.

We configured encrypted DNS at the system level and tested apps adopting HTTP(S)DNS resolvers. Do53 is successfully encrypted, but HTTP(S)DNS remains, exposing some sensitive data. We suggest that apps offer resolver choices similar to the secure DNS setting of PC browsers and adhere to users' privacy preferences.

B. Applications

Robustness. A common solution to deal with resolver failures in the traditional resolution scenario is to switch to other resolvers by changing the DNS configuration. However, this option is not available for apps that rely on the internal HTTP(S)DNS resolver. If the HTTP(S)DNS service fails, the app will not function properly and there will be no alternative solutions. Therefore, app developers should implement a fallback mechanism to handle service failures.

Fingerprinting Attack. The use of dispersed self-use resolvers may increase the vulnerability of apps to fingerprinting attacks. The HTTP(S)DNS resolvers are unique for each app and can be used as an identifying feature for classification purposes. Furthermore, the metadata and the plaintext in HTTPDNS can enhance the possibility of distinguishing one app from another.

C. ISPs

Evading Intrusion Detection. In some countries, ISPs are required by law to monitor and filter DNS queries to protect users from malicious apps or websites. However, HTTP(S)DNS resolvers are more difficult to detect in the wild because they have multiple implementations and are flexible. Malicious apps can use their own HTTP(S)DNS resolvers to avoid detection. Even if they are detected, they can quickly change and adapt.

D. The DNS Ecosystem

Application-level Autonomy. The use of the HTTP(S) protocol provides more freedom for applications in DNS resolution. Developers exploit the features of HTTP to make the protocol more than an HTTP tunnel. A fair proportion of apps in our measurement have adopted self-defined HTTP(S)DNS resolvers. The name resolution methods are no longer uniform. This suggests that application-level autonomy from the existing DNS infrastructure is emerging [17], and will have more influence on the name resolution.

Fragmentation on the DNS namespace. The non-standard protocol and self-use resolver put the name resolution process under the direct control of the application. We observed that some domain names were successfully resolved by HTTP(S)DNS but did not have the corresponding records when resolved by Do53. We suspect that the self-use resolver maintains a private set of RRs. Applications can now customize and augment the namespace, and their implementations may lead to fragmentation on the DNS namespace [5].

VII. Related Work

Recent studies typically focus on the IETF standard DoH protocol. We review these works on discovering DoH resolvers and analyzing DoH security and privacy issues.

Discovering DoH Resolvers. Previous studies have mainly focused on the server-side, using various methods to detect the adoption of DoH resolvers. Lu et al. [18] filtered URL patterns of DoH from a large-scale URL dataset and manually checked their availability. Böttger et al. [19] assessed the list of DoH servers maintained by the curl project. They found that DoH resolvers provided flexible patterns, which could respond to different URL paths. Deccio et al. [20] conducted a partial measurement and evaluated DoH adoption in open resolvers by active scanning, using the standard URI template introduced by RFC 8484. García et al. [21] developed a Nmap Lua script to find DoH resolvers, They tested for both HTTP/1.1 and HTTP/2, and three alternative methods based on HTTP *GET* and *POST* requests.

Analyzing DoH Security & Privacy Issues. Although DoH is encrypted, it still poses some security and privacy issues. Huang [22] found that DoH could be vulnerable to the downgrade attack, which exposes DNS contents to attackers. They examined six browsers with four attack vectors and found all combinations resulted in successful attacks. Another potential attack on DoH is fingerprinting attack. Siby et al. [23] treated encrypted DNS fingerprinting as a supervised learning problem and extracted features consisting of n-grams of TLS packet lengths in a trace. Vekshin et al. [24] extracted statistical and timing features to detect DoH traffic and chose several ensemble learning algorithms to validate effectiveness. Bushart et al. [25] analyzed padding strategies in encrypted DNS. They extracted statistical features of sizes and timing, used k-Nearest Neighbors as the classifier, and showed that the involved padding methods were insufficient.

VIII. Conclusion

This paper investigates the application-level name resolution in both Android and iOS apps and conducts an empirical study on the adoption and usage patterns of HTTP(S)DNS. By analyzing 25 high-profile apps, we find that non-standard HTTP(S)DNS is more prevalent and that apps tend to build their own resolvers. Moreover, we observe HTTP(S)DNS implementations in apps are diverse and scattered. Our findings complement the existing knowledge of the DNS ecosystem from the client-side perspective and suggest that the DNS ecosystem is undergoing changes. We discuss the possible impacts of these changes and highlight some potential security implications.

IX. Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments, and hard work of MESALab (www.mesalab.cn). This work is supported by the Strategic Priority Research Program of the Chinese Academy of Sciences with No. XDC02030000.

References

[1] V. Bertola, "Recommendations for DNS privacy client applications." [Online]. Available: https://datatracker.ietf.org/doc/html/draft-bertola-bcp-doh-clients-01

[2] P. Hoffman and P. McManus, "DNS queries over HTTPS (DoH)," p. RFC8484.

[3] JSON API for DNS over HTTPS (DoH) | public DNS. [Online]. Available: https://developers.google.com/speed/public-dns/docs/doh/json

[4] Using JSON · cloudflare 1.1.1.1 docs. [Online]. Available: https://developers.cloudflare.com/1.1.1.1/encrypted-dns/dns-over-https/make-api-requests/dns-json

[5] G. Huston. Where is the DNS heading? [Online]. Available: https://blog.apnic.net/2020/06/18/where-is-the-dns-heading/

[6] Baidu httpdns. [Online]. Available: https://cloud.baidu.com/product/httpdns.html

[7] aliyun httpdns. [Online]. Available: https://www.aliyun.com/product/httpdns

[8] Httpdns scheduling. [Online]. Available: https://intl.cloud.tencent.com/document/product/267/31568

[9] DNS-over-HTTPS (DoH) — Public DNS. [Online]. Available: https://developers.google.com/speed/public-dns/docs/doh

[10] DNS over HTTPS · Cloudflare 1.1.1.1 docs. [Online]. Available: https://developers.cloudflare.com/1.1.1.1/encryption/dns-over-https/

[11] DoH with Quad9 DNS Servers. [Online]. Available: https://www.quad9.net/news/blog/doh-with-quad9-dns-servers/

[12] About open data | rapid7. [Online]. Available: https://opendata.rapid7.com/about

[13] C. Brubaker. Changes to trusted certificate authorities in android nougat. [Online]. Available: https://android-developers.googleblog.com/2016/07/changes-to-trusted-certificate.html

[14] Q. Huang, D. Chang, and Z. Li, "A comprehensive study of DNS-over-HTTPS downgrade attack," in *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*. USENIX Association, Aug. 2020.

[15] Mmtls: Introduction of tlsv1.3 based tecent security communication prtocol. [Online]. Available: https://github.com/WeMobileDev/article/blob/master/SUMMARY.md

[16] Routeviews prefix to AS mappings dataset (pfx2as) for IPv4 and IPv6. [Online]. Available: https://www.caida.org/catalog/datasets/routeviews-prefix2as/

[17] G. Huston. (2020) Improving the privacy of DNS and DoH with oblivion. [Online]. Available: https://blog.apnic.net/2020/12/16/improving-the-privacy-of-dns-and-doh-with-oblivion/

[18] C. Lu, B. Liu, Z. Li, S. Hao, H. Duan, M. Zhang, C. Leng, Y. Liu, Z. Zhang, and J. Wu, "An end-to-end, large-scale measurement of DNS-over-encryption: How far have we come?" in *Proceedings of the Internet Measurement Conference*, ser. IMC '19. Association for Computing Machinery, pp. 22–35.

[19] T. Böttger, F. Cuadrado, G. Antichi, E. L. Fernandes, G. Tyson, I. Castro, and S. Uhlig, "An empirical study of the cost of DNS-over-HTTPS," in *Proceedings of the Internet Measurement Conference*, ser. IMC '19. Association for Computing Machinery, pp. 15–21.

[20] C. Deccio and J. Davis, "DNS privacy in practice and preparation," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, ser. CoNEXT '19. Association for Computing Machinery, pp. 138–143.

[21] S. García, K. Hynek, D. Vekshin, T. Čejka, and A. Wasicek, "Large Scale Measurement on the Adoption of Encrypted DNS," *arXiv:2107.04436 [cs]*, Jul. 2021.

[22] Q. Huang, D. Chang, and Z. Li, "A Comprehensive Study of DNS-over-HTTPS Downgrade Attack," in *10th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 20)*, 2020.

[23] S. Siby, M. Juárez, C. Díaz, N. Vallina-Rodriguez, and C. Troncoso, "Encrypted DNS -> privacy? A traffic analysis perspective," in *27th Annual Network and Distributed System Security Symposium, NDSS 2020*. The Internet Society, 2020.

[24] D. Vekshin, K. Hynek, and T. Cejka, "DoH Insight: Detecting DNS over HTTPS by machine learning," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ser. ARES '20. Association for Computing Machinery, Aug. 2020, pp. 1–8.

[25] J. Bushart and C. Rossow, "Padding Ain't Enough: Assessing the Privacy Guarantees of Encrypted {DNS}," in *10th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 20)*, 2020.