Dmap: Automating Domain Name Ecosystem Measurements and Applications

Maarten Wullink, Giovane C. M. Moura, and Cristian Hesselman
SIDN Labs
Arnhem, The Netherlands
firstname.lastname@sidn.nl

Abstract—Behind each Internet domain name, there is a set of entities/companies responsible for delivering the various services associated with it, such as Web hosting and e-mail. Together, they form what we refer to as DNS ecosystem. Currently, there is no single measurement tool designed to measure this ecosystem altogether. As a result, researchers that aim at analyzing (parts of) this ecosystem often have to spend significant amounts of time preparing and executing the multiple application measurements and post-processing their heterogeneous raw datasets. Given that time is a scare resource, this complexity diverts researcher’s time from actual analysis, ultimately limiting how far many studies go. To help researchers facing this situation, we present Dmap, an active measurement application that reduces the complexity of executing both measurements and analysis. It does so by (i) automating the crawling of several application protocols (DNS, HTTP, TLS/SSL, SMTP, both over IPv4 and IPv6) and (ii) storing the results into a relational data base, enabling researchers to quickly perform hypothesis tests within interactive response times using SQL. Dmap current version has 40 classifiers that generate 166 derived features (e.g., CMS detection, page language), which can be used by researchers and operators to build applications and services. We present an evaluation of Dmap and show three applications that it can be used for, such as profiling the Alexa 1 million domains. We use Dmap at SIDN (.nl registry) for research on the .nl zone and make it open-source for researchers.

I. INTRODUCTION

The Internet Domain Naming System (DNS) provides a global hierarchical naming space in which labels are used for hosts, services, applications, and networks on the Internet [1]. It comprises one of the core services on the Internet [2].

Associated with each domain name, there is a diverse set of entities/companies providing various services/functionalities, forming what we refer to in this paper as DNS ecosystem. To illustrate it, consider the domain wikipedia.org in Figure 1. To register a domain, there are typically three entities involved: registrant, registrar, and registry. A registrant (person or company) chooses a registrar (e.g., GoDaddy) which is accredited by the top-level domain (TLD) registry (Public Interest Registry for .org) to register the domain (wikipedia.org) on behalf of the registrant. After that, a registrant sets the authoritative name servers [3] ({ns0,nls1,nls2}.wikimedia.org) to answer queries to the domain. Each of these authoritative name servers in turn can be run by different DNS providers (e.g., Dyn, Amazon Route53) on different networks/locations.

After that, the registrant can associate diverse services with the domain, such as web, and e-mail (using DNS MX records [4]). Each of these services can also be provided by different companies (e.g., GMail or Yahoo for e-mail).

This example illustrates the complexity of the DNS ecosystem. Currently, there is no open-source measurement tool that measures this ecosystem altogether. Instead, there is a set of application-specific tools that produce heterogeneous datasets (§II). As a consequence, researchers often spend valuable time on (i) both coordinating measurements (due to heterogeneous tools) and (ii) preparing the collected data (handling different formats), instead of spending time on actual research questions. Ultimately, this complexity operates as veiled limiting factor in many measurements studies, by consuming time that otherwise could be spent on data analysis and hypothesis tests.

To help researchers and operators facing this scenario, we introduce DNS Ecosystem Mapper (Dmap, available at [5]), an open-source, scalable, modular, and distributed RESTful web application [6] that reduces the complexity in both carrying the measurements related to the domain name ecosystem and analyzing the collected data. It does that by (§III) (i) automating the crawling of various application layer protocols (DNS, HTTP, SMTP, and TLS, for both IPv4 and IPv6) for any given domain name and by employing a unified relational database model, which (ii) enables researchers to perform hypothesis tests using SQL syntax, obtaining results within interactive response times. Besides that, Dmap goes beyond current tools by enriching the raw measurements with derived features, which is done with the use of 40 classifiers that,
among others tasks, detect the language of a web page and determine if and which content management system (CMS) are deployed. Ultimately Dmap produces up to 166 features for a given domain name.

At SIDN (nl top-level domain (TLD) operator), we employ Dmap to perform periodical full zone scans (~ 5.8 million domains) to support projects that focus both on improving security and stability of the DNS [7]. We assess its performance and precision using a production DNS zone (.nl, §IV).

By significantly reducing the complexity associated with the execution of measurements and data engineering, Dmap frees researchers to spend more time in the data analysis and focus on research questions. We show three applications of Dmap and show how data analysis can be easily done with SQL (§V).

Finally, we make Dmap open for researchers [5].

II. Yet Another Scanner?

Over the last few years, the Internet measurement community has been benefiting from both fast scanners and open measurement data repositories. So one may wonder: do we need yet another scanner?

To answer this question, we first lay out the requirements for Dmap and then discuss how current tools/repositories do not fulfill them.

A. Requirements

The main requirement for Dmap is to reduce the complexity associated with both execution and analysis of measurements related to the domain name ecosystem (Figure 1). To achieve that, it needs to fulfill the following sub-requirements:

1) **Domain-centric**: Dmap should take as input a list of domains, as defined by its users, and not a list of IP addresses. It can be used to perform scans on entire DNS zones or a subset of them.

2) **Automate the crawling of multiple protocols**: It should crawl various protocols (DNS, HTTP, SMTP, and TLS, currently), and generate a screenshot of the main page of a domain, if exists.

3) **IPv4 and IPv6 support**: it should measure actively the application protocols from #2 using both IPv4 and IPv6, and store the results separately.

4) **Completeness**: Dmap should aim for completeness of results, by having fail-safe mechanisms that allow for retrial of measurements in case of transient failures. Users should be able to set timeout and retry options for each measurement.

5) **Derived features using classifiers**: Dmap should produce derived features from raw datasets. Those features are those that require extra processing, such as identification of CMSes and languages identification on Web sites, among others.

6) **SQL-based unified data model**: features from the raw measurement datasets should be extracted and then stored in unified SQL-based data model, in an external relational database management system (RDBMS). This model enables researchers and engineers to analyze large volume of data with interactive response times (seconds to minutes), which is essential for hypothesis tests and design of new applications [8], [9].

7) **Modular**: users should be able to control which application protocols should be measured (#2).

8) **Distributed**: Dmap architecture should be distributed for scalability.

9) **Open-source**: the tool should be open-source for researchers, so users have a complete understanding on how measurements are carried.

B. Current Tools and Services

Currently, there are various measurement tools that can be used to carry out Internet measurements. However, none of them carry multi-application measurements. As a consequence, users have to use distinct applications for different protocols, and some do not support IPv6. For example, Masscan [10] and Zmap [11] are tools that have decreased significantly the time required for perform IPv4 port scans, while ZMap subtools [12] addressed specific applications, such as HTTP and TLS.

While they have significant value, these tools are fragmentation: for each type of measurement, one needs to employ a specific tool, increasing the complexity. And each of them have of them have a distinct data format. As such, it does not fulfill sub-requirements #2 and #6.

Moreover, Masscan and ZMap are designed with a different goal: scan the entire IPv4 space. Dmap, on the other hand, is designed to scan domain names. The domain name space is different from the IP address space, and given the prevalence of web shared hosting [13] (in which many sites are hosted on the same servers), IP-wide scans cannot be used to enumerate (and measure) the domain name space, since such tools cannot determine which are the domains present in shared hosting services.

Other services such as DomainTools [14] provide similar features as Dmap. However, they are paid services and not open-source for researchers – violating requirement #9.

C. Public Data Repositories

Besides open measurement tools, Researchers have also available several large public measurement data repositories. CAIDA [15], USC/ISI’s ANT [16], scans.io [17] and censys.io [18] are among the most popular ones.

Similarly to the existing tools, these datasets provide valuable data to the research community. However, they may not cover all cases. For example, at SIDN, we periodically scan our .nl zone at a frequency we find necessary. Other researchers, for example, may use Dmap to scan other publicly available zones [19] at their will. Thus, these repositories, of course, cannot cope with the user’s specifics requirements, not fulfilling requirement #1. Besides, they also come in different data formats (#6) and may not support IPv6 (#3).

III. Dmap Design Principles

Different from most measurements tools, which are stand-alone, Dmap has a distributed client/server architecture in form
of a Representational state transfer (RESTful) [6] web service developed in Java. uses Spring Boot [20], which is a Java framework that contains a series of libraries/tools that abstracts part of the complexity in writing RESTful applications. With Dmap, users have full control of what, when, and what protocols to measure.

Besides that, Dmap uses other third-party libraries (e.g., Maxmind GeoIP database [21] for annotating IP addresses, PhantomJS [22] for screenshots, the Java VM, and PostgreSQL [23] to store the final results). Even though Dmap has a distributed architecture, it can run on a single server as a .jar file.

Figure 2 shows the architecture of Dmap. It is divided in three main modules: Data preparation (A), Crawlers (B), and Classifiers (C). Next, we cover these in more details.

A. Data Preparation and Control (A)

The first module of Dmap consists of preparing the data to be used in the scans. Note that Dmap can also be used to scan individual domains using a web interface (also generating JSON output) but in this section we focus on how it performs scans in bulk, i.e., on thousands or million domain names.

First, a user invokes the Crawler controller via an URI (step 1 on Figure 2), using the RESTful API. This triggers the Crawler service (2), which has two main functions: read the contents of a .csv file containing the list of domains to be scanned and storing it into DomainsDB, which is a PostgreSQL database.

Then, the same Crawler service reads \( n \) random domains from Domain DB and inserts these into the Input Queue (3), which in fact is a in-memory database used by the crawlers to start the scan. We employ Hazelcast [24] as our Input Queue, which is a lightweight open-source in-memory database.

The Crawler service also monitors the size of Input Queue as the scans take place, and once it drops below a configurable threshold, it then fetches more random domains from Domains DB and inserts them into Input Queue. We choose to place \( n \) random domains (e.g., 15,000) each time at Input Queue instead of all of them in order to Dmap’s reduce memory footprint and to minimize possible data loss during scans in case of application errors. This allow us to re-start the scan from where it stopped just by looking at the domain’s states stored in DomainsDB (§III-D).

B. Crawlers Module (B)

The crawler module (B) of Dmap is responsible for carrying out the actual application level measurements. We first describe how they work in broad terms, and then cover the specifics of each crawler.

After storing \( n \) random domains in Input Queue (step 3), the application starts \( x \) Crawler Threads (step 4 in Figure 2, configurable by the user). Each thread then reads one domain from the Input Queue and executes the Crawler Pipeline (5) for the domain.

The Crawler Pipeline defines the sequences of measurements that will be executed. Currently, it is composed of ten stages: eight designed to measure application protocols (DNS, HTTP, TLS, and SMTP); another stage used to capture a screenshot of a website and one stage (REG) to retrieve domain name registration information (typically internal to TLD registries, but it could be extended to support third-party APIs).

Each individual crawler of the pipeline can also be optionally activated/deactivate for each measurement. For example, a user interested only in DNS information can disable all other crawlers, thus reducing the total measurement time.

Dmap keeps the state of each measurement. To do that, its Crawler Service (2) keeps in DomainsDB metadata that indicate which domains have been crawled, which need to be retried, which protocols need to still to be crawled.

After performing its measurements, each crawler in the pipeline stores its raw measurement results into the the Raw Result Queue (step 6), which is a Java queue implementation. We refer to this queue as “raw” since it contains different data formats, such as html (for HTTP measurements). By default, Dmap does not store the raw measurement results – only results as specified in our data model (§III-F) (we are currently working on implementing a module to export this raw datasets, so it can be further analyzed using other solutions such as Elasticsearch [25]).

C. Crawlers Details

We have developed each crawler of the pipeline from scratch, using a series of external third-party libraries. The reasons for doing this is that it allows us to have full control over the measurement data and processing, enabling us to extract whatever features we needed. Next we discuss each crawler.

1) DNS crawler: the DNS is used to store various types of records. For example, A [1] and AAAA [26] records are used to stores IP addresses (IPv4 and IPv6 respectively, the “classic” name to IP mapping). However, there are other record types too: MX records [4] specify mail exchange for the domain (e.g., smtp.example.nl), while SOA specifies the start
of the zone authority. TXT records [4], on the other hand, hold descriptive data, and are use in DMARC [27] and DKIM [28] protocols. Our crawler downloads all these records, which are later processed by classifiers that annotate them with other features.

2) HTTP crawler: we use Apache’s HttpClient [29] library, which provides an implementation of the HTTP client protocol. We have implemented, however, the logic behind the crawler, and handled a series of issues. For example, if a domain redirects to another domain or subdomain, our crawler will follow to the redirect page (up to a certain number of redirects configured by the user). Also it is able to handle many corner cases such as websites that continue streaming large amounts of data, by using timeouts and maximum bytes allowed to download.

3) TLS crawler: The crawler attempts to establish a HTTPS connection to the domain name, and download its X.509 certificate [30] (which is not part of the protocol). We disable all Java security controls, to prevent that exceptions are raised occur during this phase, so we are able to determine if there is an error with a certificate, what error it is (expired, self-signed, etc.).

4) SMTP crawler: the SMTP crawler crawls all configured mail servers (given by MX records) and verifies if they support StartTLS (opportunistic TLS) [31]. The crawler will try to create a connection to each mail server IP address (over both IPv4 and IPv6).

5) Screenshot crawler: it creates a screenshot of the main web page and uses PhantomJS for that. This is a relatively expensive operation because all the website resources (e.g., HTML, JavaScript, Images) must be downloaded and then the page has to be rendered. Therefore, we disable this crawler by default to speed up large zone scans. However, users can activate as they wish.

6) REG crawler: we use it internally at our .nl registry, and we retrieve information such as registration date and registrar. Our open-source version has these features disabled, since they are registry only. This crawler could be extended to connect to third-party APIs or whois services to extract more features.

D. Fail-safe Mechanisms

Measurements results may be affected by transient errors [32], such as connectivity issues and packet loss. To handle such errors, we have built into Dmap various fail-safe mechanisms.

First, for each domain $d$, we keep in the DomainsDB metadata about the stage in which the domain is in the pipeline. If one of the crawlers in the pipeline fails, Dmap stores extra information associated with that — such as when it failed, and the respective error. It also keeps a counter for number of retries and a time for the next time it should be retried (both parameters configurable by the user).

We also handle different exceptions for each type of crawler. For example, for the DNS crawler, we do not retry if a domain name resolution was answered with an NXDOMAIN [1], which indicates the requested domain does not exist. If there is a timeout, however, Dmap retries again to crawl the domain after a configure time interval.

E. Classifiers Module (C)

Classifiers have two roles on Dmap: extract the required features from the raw measurement data and enrich this data with other derived features. Such derived features is one of the biggest differences between Dmap and other tools: it generates by default a richer set of features that would require researchers to manually implement them.

Table I shows 6 of these classifiers and their respective features. For example, ServerClass attempts to perform OS and Web server identification, while TLSCert reads the X.509 certificates, extract its type, CA, among others. Many domains are also “parked” [33], i.e., a domain that is not developed and usually used in automatically generated advertisement.

The CMS classifier attempts to identify a content management system (CMS), such as Joomla, is in use on a website, besides detecting the presence of forums or shopping carts (used to detect online stores). To do that, we have manually inspected various CMSes and produced matching patterns for 42 of them. For example, to identify Wordpress, we look into the generator tag found in the HTML code of the main page and perform substring match with Wordpress and WPML. In addition, we also search for internal links with wp-content and wp-includes on their paths (these matching patterns need to be updated as new versions of CMSes are released).

Table II shows 65 features produced by HTTP classifiers, from the raw HTML code and its metadata. Features such as httpStatus, total number of links and internal links (htmlLinksInt, htmlLinksAll), and site language (pageLan) can be directly used by researchers to build classifiers to detect, for example, malicious web sites.

Currently, there are 40 classifiers on Dmap. Each classifier, in turn, is associated to one crawler type only. For example, the Parking classifier (Table IV) process the raw html data from the HTTP crawler. Users can also develop new classifiers or modify existing ones. Users cannot, however, disable individual features – they can either enable or disable classifiers (§III-C), but once a crawler is activated, their respective classifiers will generate the entire set of features.

Ultimately, the classifiers store the results into Results DB (9), an external database upon which analysts can carry out their hypothesis tests and build other applications. We cur-
TABLE II
65 HTTP IPv4 features produced by Dmap for en.wikipedia.org.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>domainname</td>
<td>&quot;wikipedia.org&quot;</td>
</tr>
<tr>
<td>crawlRun</td>
<td>764</td>
</tr>
<tr>
<td>ipVersion</td>
<td>4</td>
</tr>
<tr>
<td>url</td>
<td>null</td>
</tr>
<tr>
<td>crawlName</td>
<td>&quot;wikipedia.org&quot;</td>
</tr>
<tr>
<td>crawlUrl</td>
<td>&quot;<a href="https://www.wikipedia.org/">https://www.wikipedia.org/</a>&quot;</td>
</tr>
<tr>
<td>crawlDomain</td>
<td>&quot;wikipedia.org&quot;</td>
</tr>
<tr>
<td>crawlDate</td>
<td>&quot;2018-05-15 T09:28:48.642+0000&quot;</td>
</tr>
<tr>
<td>crawlStatus</td>
<td>0</td>
</tr>
<tr>
<td>crawlPages</td>
<td>2</td>
</tr>
<tr>
<td>crawlRetries</td>
<td>0</td>
</tr>
<tr>
<td>networkLoadTime</td>
<td>24</td>
</tr>
<tr>
<td>networkLoadTimeAll</td>
<td>24</td>
</tr>
</tbody>
</table>

This table shows the 65 HTTP IPv4 features produced by Dmap for en.wikipedia.org. For example, the feature `domainname` has a value of "wikipedia.org", and the `networkLoadTimeAll` feature has a value of 24.

Fig. 3. HTTP crawler result codes

To evaluate Dmap, we have carried out a full zone on our .nl TLD zone on September 3rd, 2017 from our corporate network. In total, 5,766,118 domains were used as input. We ran it on a single virtual machine with 8x 1.4GHz CPU cores with 32GB of RAM, and 1 Gbps shared line. We activated 6 crawlers (screenshot was disabled), with 300 Crawler threads and 50 Classifier threads (Figure 2).

A. Precision/Completeness

To determine how precise and complete Dmap is, we compare its results for DNS services against OpenIntel [38], a project that daily perform DNS scans of various DNS zones and that uses a different vantage point from ours.

On September 3rd, 2017, after scanning all .nl domains for DNS records, OpenIntel listed 913,511 domains that had a AAAA record associated to it, i.e., a IPv6 address (~15.8% of the total – not every domain needs to have an IPv6 AAAA record). Dmap, in turn, found that 913,593 .nl domains had AAAA records – a very similar figure. The small difference (82 domains) can be explained by transient changes in these domains setups while being measured.

For DNS zones, such as .nl, we can expect that not all domains have a Web page (some not used for Web, others not depeleved yet). We then show the results for the HTTP crawler over the .nl zone. As discussed in §III-B, this crawler downloads the main page associated with a domain (obtained from its A or AAAA records). Figure 3 shows the results of the HTTP crawlers. Out of 5,766,118 domains, 4,756,943 (82.49%) had an active Web page on IPv4 (OK on x axis), and 811,425 had an IPv6 page (14.07%) (note that we found that 914,593 domains had an IPv6 AAAA record associated to it, but only 811,425 had a Web server running on this address).

Fail-safe mechanism: as discussed in §III-B, we have built a fail-safe mechanism in each crawler. To evaluate its behavior, we analyze how many times the HTTP-crawler had perform
before it could connect to the domains with available pages. We found that only 40 domains for IPv6 required at least one connection retrial (after the first failed), and only 38 for IPv4. After that, if they did not responded, we collected the error codes shown in Figure 3. As such, most connections either work for the first time or not worked at all.

B. Performance

Performance is an important feature of any Internet measurement application. In our single VM setup, Dmap crawled, on average, 11.34 domain per second (980,000/day), with all crawlers activated except for the Screenshot crawler. However, Dmap is designed to scale horizontally in terms of performance, i.e., adding more crawling nodes (part B and C on Figure 2) can linearly improve the performance whenever needed. Also, depending on the user’s needs, some crawlers from the pipeline can be disabled, thus reducing the total measurement time.

C. Comparison with other tools

Given the lack of comparable open-source tools that produce the same number of features (§II), it is hard to draw a direct comparison with Dmap.

Still, compared to other stateless IPv4 scanners such as Masscan and ZMap (which produces far fewer features), Dmap performance may seem humble. However, there are reasons for that: first, the number of features (up to 166) generated, which required 5 crawlers that performed tasks such as resolving domain names, establishing TLS sessions, retrieving certificates, HTTP connections over both IPv4 and IPv6, and run a series of classifiers on the data. Therefore, there is intrinsic a trade-off between number of features and performance.

This performance difference between IPv4 scanners and Dmap is less critical also given the differences in scan space. For example, for second-level domains (e.g., example.nl and not portal.example.nl), the .com zone, which is the biggest, has \( \sim 130 \) million second-level domain names. Comparatively, the IPv4 address space has more than 4 billion addresses and the IPv6 has \( 3.4 \times 10^{38} \) addresses.

V. APPLICATIONS

In this section, we present three applications for which we have used Dmap. We do not intend to present a comprehensive list of applications; rather we show how easy Dmap can be used for different use cases.

A. Profiling Alexa 1 Million

The Alexa 1 million domains list which ranks sites based on their toolbar users [39, 40]. We crawled this list of domains with Dmap on February 21st, 2018 and present here a descriptive analysis of the results obtained. We also make available the resulting dataset (PostgreSQL database) and SQL code for this analysis available in [5].

Table III shows a profile of the Alexa 1M domains divided by protocol, for both IPv4 and IPv6. Let’s start with the DNS protocol: the row # Domains OK show the number of domains that had at least one of its authoritative serves (defined by NS records in DNS) with IPv6 (defined by a AAAA record associated to a NS record).

Then, we see that \( \sim 289K \) authoritative servers (NS records, such as ns.google.com) were shared among 1M domains. These authoritative, in turned, were mapped to \( \sim 210K \) IPv4 addresses, distributed over \( \sim 18K \) Autonomous Systems (ASes). We select two findings from this data:

---

**Table III**

**ALEXA 1 MILLION PROFILING**

<table>
<thead>
<tr>
<th>DNS</th>
<th>IPv4</th>
<th>IPv6</th>
<th>IPv6/IPv4</th>
</tr>
</thead>
<tbody>
<tr>
<td># Domains (OK)</td>
<td>972,155</td>
<td>153,485</td>
<td>0.16</td>
</tr>
<tr>
<td># Unique NSes</td>
<td>289,014</td>
<td>26,127</td>
<td>0.09</td>
</tr>
<tr>
<td># Unique IP</td>
<td>210,650</td>
<td>19,754</td>
<td>0.09</td>
</tr>
<tr>
<td># Unique ASes</td>
<td>18,418</td>
<td>3,178</td>
<td>0.17</td>
</tr>
<tr>
<td># CDN Cloudflare</td>
<td>117,538</td>
<td>115,396</td>
<td>0.98</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HTTP</th>
<th>IPv4</th>
<th>IPv6</th>
<th>IPv6/IPv4</th>
</tr>
</thead>
<tbody>
<tr>
<td># Domains (OK)</td>
<td>968,338</td>
<td>153,485</td>
<td>0.16</td>
</tr>
<tr>
<td># HTML 5</td>
<td>681,757</td>
<td>116,066</td>
<td>0.17</td>
</tr>
<tr>
<td>Bytes (median)</td>
<td>53,889</td>
<td>64,735</td>
<td>1.20</td>
</tr>
<tr>
<td>Internal links (median)</td>
<td>7</td>
<td>8</td>
<td>1.14</td>
</tr>
<tr>
<td>Cookies (median)</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TLS</th>
<th>IPv4</th>
<th>IPv6</th>
<th>IPv6/IPv4</th>
</tr>
</thead>
<tbody>
<tr>
<td># Domains (OK)</td>
<td>772,455</td>
<td>129,443</td>
<td>0.17</td>
</tr>
<tr>
<td># Let’s Encrypt</td>
<td>165,526</td>
<td>10,113</td>
<td>0.06</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SMTP</th>
<th>IPv4</th>
<th>IPv6</th>
<th>IPv6/IPv4</th>
</tr>
</thead>
<tbody>
<tr>
<td># Domains (OK)</td>
<td>843,126</td>
<td>190,736</td>
<td>0.23</td>
</tr>
<tr>
<td># Unique SMTP</td>
<td>501,848</td>
<td>24,311</td>
<td>0.05</td>
</tr>
<tr>
<td># Unique IP</td>
<td>286,504</td>
<td>10,113</td>
<td>0.04</td>
</tr>
<tr>
<td># Unique StartTLS</td>
<td>302,871</td>
<td>8,016</td>
<td>0.03</td>
</tr>
</tbody>
</table>

---

77.2% of domains deploy Web encryption (IPv4), and 1 in 5 now use Let’s Encrypt: To enable web encryption, a Web site needs an X.509 certificate [30], which is issued by a Certificate Authorities (CAs). We found that 77.2% of all domains now have Web encryption over IPv4. Among the available CAs, Let’s Encrypt [42] foundation has drawn significant attention over the last years since it was the first CA to provide both free X.509 certificates and automated software for that. In a previous study, we evaluated the first year of Let’s Encrypt in terms of certificate issuance [43] for all domains and found that Let’s Encrypt grew very fast in the first year. Now, we can see how many domains of the 1M list deploy Let’s encrypt certificates on second-level domain. As can be seen, for IPv4, 21.4% of domains that support SSL/TLS over HTTP employ Let’s Encrypt.
B. Longitudinal Studies on Web Encryption Adoption

Dmap can be used to crawl periodically the same domains and support longitudinal studies of changes in a domain name set. In this section, we show how it enables to characterize and determine how much Web encryption adoption has changed for the .nl zone.

With other tools, this rather straightforward question may take researchers significant amounts of time to both measure and analyze it. With Dmap, a researcher can used the SQL query showed in Listing 1 to retrieve the results. In our setup, it took 8s to run this query.

Listing 1. SQL query to analyze web encryption on the .nl zone.

```sql
SELECT count(domainname) FROM crawl_result_tls
WHERE
1. Variable ordering where, self-signed and EV certificates are the primary choice.
2. The .nl zone scan. WordPress is the most popular CMS, and Apache is the most popular web server. Dmap is the first open-source active measurements application that focus on crawling various applications, extracting a series of features from the raw datasets and producing a derived features with its classifiers, and storing the results in a relational data base. We have previously discussed existing measurement tools in §II.

There are other works that are similar but not directly related. For example, OpenIntel [38] focus on daily crawls of DNS records of multiple zones (thus comparable to our DNS crawler – §III-C). However, it does not crawl applications other than DNS. Dmap, besides DNS, also crawls HTTP, TLS, SMTP protocols as well, enriching the raw data with a series of derived features with the aid of its classifiers.

Another similar work is our previous open-source ENTRADA, which is data streaming warehousing for authoritative DNS traffic. However, ENTRADA is designed to collect and store passive measurement data, while Dmap is an active measurement tool. However, both tools provide a SQL interface for data analysis.

VII. CONCLUSION AND FUTURE WORK

Internet measurements can be complex and demand significant efforts in planning, executing, and analyzing the collected data [32]. All this “heavy lifting” must be carefully executed before a researcher can begin to answer their research questions. This is still the case for many researchers, especially when public data repositories and tools do not fulfill the

### TABLE IV

<table>
<thead>
<tr>
<th>Types of Certificates</th>
<th>20170903</th>
<th>20180201</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zone size</td>
<td>5,766,118</td>
<td>5,801,191</td>
</tr>
<tr>
<td>TLS/SSL</td>
<td>2,595,281</td>
<td>2,674,877</td>
</tr>
<tr>
<td>Unknown</td>
<td>382 (0.01%)</td>
<td>407 (0.01%)</td>
</tr>
<tr>
<td>self-signed</td>
<td>833,844 (54.05%)</td>
<td>749,033 (28.01%)</td>
</tr>
</tbody>
</table>
| DV                    | 1,406,072 (54.17%)| 1,618,575 (60.5%)
| OV                    | 269,284 (10.37%)| 272,318 (10.18%)
| EV                    | 35,699 (1.37%)| 34,544 (1.29%)

Let’s Encrypt (DV) deployed certificates share

| Let’s Encrypt (%)DV | 366,623 (14.12%)| 523,029 (32.31%)|

### TABLE V

<table>
<thead>
<tr>
<th>CMS</th>
<th>Total (%)</th>
<th>Web Server</th>
<th>Total (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wordpress</td>
<td>848,083 (73.84%)</td>
<td>Apache</td>
<td>3,397,930 (77.05%)</td>
</tr>
<tr>
<td>Joomla</td>
<td>99,865 (8.69%)</td>
<td>nginx</td>
<td>696,697 (15.79%)</td>
</tr>
<tr>
<td>Drupal</td>
<td>40,798 (3.55%)</td>
<td>MS ISS</td>
<td>249,367 (5.65%)</td>
</tr>
<tr>
<td>Blogo</td>
<td>20,749 (1.80%)</td>
<td>cloud-nginx</td>
<td>60,530 (1.13%)</td>
</tr>
<tr>
<td>Others</td>
<td>121,232 (10.49%)</td>
<td>Others</td>
<td>905 (0.00%)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1,448,408</td>
<td><strong>Total</strong></td>
<td>4,409,666</td>
</tr>
</tbody>
</table>
researchers’ requirements. As such, this complexity is in fact a limiting factor for many studies.

In this paper, we presented Dmap, a measurement tool which its main goal is to reduce the complexity in both carrying measurements and analyzing the collected data by automating multi-application measurements and data preparation. Researchers (and DNS operators), thus, can profit from Dmap by spending their valuable time in hypothesis tests and data analysis. We make Dmap source-code and binaries available for researchers [5]. As future work, we intend to add other crawlers to Dmap, so new application layer protocols can be supported and develop new classifier to produce more derived features.

Overall, we hope Dmap can be useful for researchers and operators to build applications that help to improve both security and stability of global DNS.

Acknowledgments: We would like to thank Moritz Müller and the anonymous TMA reviewers for their valuable comments on paper drafts.

REFERENCES