FlowDNS: Correlating Netflow and DNS Streams at Scale

Aniss Maghsoudlou
aniss@mpi-inf.mpg.de
Max Planck Institute for Informatics
Saarbrücken, Germany

Oliver Gasser
oliver.gasser@mpi-inf.mpg.de
Max Planck Institute for Informatics
Saarbrücken, Germany

Ingmar Poese
ipoese@benocs.com
Benocs GmbH
Berlin, Germany

Anja Feldmann
anja@mpi-inf.mpg.de
Max Planck Institute for Informatics
Saarbrücken, Germany

ABSTRACT

Knowing customer's interests, e.g. which Video-On-Demand (VoD) or Social Network services they are using, helps telecommunication companies with better network planning to enhance the performance exactly where the customer's interests lie, and also offer the customers relevant commercial packages. However, with the increasing deployment of CDNs by different services, identification, and attribution of the traffic on network-layer information alone becomes a challenge: If multiple services are using the same CDN provider, they cannot be easily distinguished based on IP prefixes alone. Therefore, it is crucial to go beyond pure network-layer information for traffic attribution.

In this work, we leverage real-time DNS responses gathered by the clients' default DNS resolvers. Having these DNS responses and correlating them with network-layer headers, we are able to translate CDN-hosted domains to the actual services they belong to. We design a correlation system for this purpose and deploy it at a large European ISP. With our system, we can correlate an average of 81.7% of the traffic with the corresponding services, without any loss on our live data streams. Our correlation results also show that 0.5% of the daily traffic contains malformatted, spamming, or phishing domain names. Moreover, ISPs can correlate the results with their BGP information to find more details about the origin and destination of the traffic. We plan to publish our correlation software for other researchers or network operators to use.

ACM Reference Format:
Aniss Maghsoudlou, Oliver Gasser, Ingmar Poese, and Anja Feldmann. 2022. FlowDNS: Correlating Netflow and DNS Streams at Scale. In *The 18th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '22), December 6–9, 2022, Roma, Italy*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3555050.3569135

CoNEXT '22, December 6–9, 2022, Roma, Italy
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9508-3/22/12…$15.00
https://doi.org/10.1145/3555050.3569135

1 INTRODUCTION

ISPs need to know where their traffic is coming from originally not only to provide their customers with better quality, but also to better plan their network infrastructure and collaborations.

To this end, most of the service providers gather network-layer statistics of their traffic using different protocols, e.g. Netflow [7], IPFIX [2], etc. which usually include source and destination addresses, and traffic volume.[1] Network-layer headers do not contain the domain name of the services they belong to. To complicate this even more, Over-The-Top (OTT) services are nowadays adopting multi-CDN approaches [24], making the inference of the service merely on the IP address nearly impossible. Therefore, either DNS records or the application layer information are needed. Nowadays, the application layer information is oftentimes encrypted [10] and therefore, not visible to the service providers.

DNS is one of the core services to map domain name to the IP address [15, 16], and can be used to find the original source of the service. There have been several studies using machine learning techniques for domain name recognition, all of which use passive DNS records [4–6, 13, 21]. Inspecting the traffic on a large European ISP, we observe that more than 85% of the traffic is originated by CDNs. These CDNs might use one domain name for several services in different locations/times and IP addresses could also be re-used [22]. Therefore, IP address to domain name mappings change frequently in CDN-hosted domain names [19, 22], and DNS records used for such correlation should not be outdated. Thus, capturing the DNS records collected from user requests is the most suitable way of mapping domain name to IP address.

Previous studies use software-defined networking to correlate DNS responses with web traffic either in an external controller [9] or in the data plane [3, 12]. However, all these approaches introduce parsing limitations, e.g. domain names length limit, and also ignore encrypted DNS packets. This work, however, does not enforce any limitation on the DNS records, and is not affected by DNS encryption. Unlike the previous studies, we do not aim at direct policy enforcement and therefore, do not require any modification to the existing network architecture. We instead propose a system that can run on any machine receiving a flow of DNS records and network flows.

Using DNS records from the same source as the traffic gives the domain name recognition more certainty. In the meanwhile, using

[1] Throughout this work, we refer to network flow information as Netflow, which is a standardized protocol to collect IP network traffic [8].

the same sources of DNS and Netflow translates to a higher processing load since both sources need to be processed synchronously either in a real-time fashion or offline. In case the processing is to be done offline, the timestamps need to be taken into account and the two sources of data, namely Netflow and DNS records, need to be correlated in the window where the DNS record is still valid, i.e. TTL > 0. Although research has shown that in some scenarios, longer TTLs can reduce latency [17], our experiments show that 99% of the record have TTLs of less than two hours. Monitoring TTLs for every single record while correlating induces higher memory usage and lower correlation rates. Multi-level caching in DNS resolvers makes this even more complicated [23].

Correlating two live sources of data, each carrying thousands or millions of records per second, also requiring keeping some of the records for later use, is not trivial. Doing so with the standard database queries for an hour of data, takes tens of hours, making this correlation impossible to be done in real-time. Therefore, we propose FlowDNS, a system to correlate Network-layer headers and DNS records in real-time. Our work consists of three main contributions:

- We design, build, and deploy a system for real-time DNS-Netflow Correlation called FlowDNS in a large European ISP. In Section 3, we go over the system's building blocks, and in Section 4, we show that we can correlate 81.7% of the data.
- Using the correlated data in our deployment of FlowDNS, we identify the traffic using malformatted, spam and phishing domains in Section 5. We observe that 0.5% of the daily traffic volume uses either malformatted or spam/phish domain names.
- Finally, we formulate our learned lessons in building a real-time DNS-Netflow Correlation system in Section 6.

2 DATA OVERVIEW

We use flow data and DNS traffic from a Large European ISP. The flow data is in Netflow format, and we receive both Netflow and DNS traffic as live streams:

- **DNS streams**: A set of DNS *cache misses* gathered from different customer resolvers. For load-balancing purposes, the data is already divided into 2 different streams, carrying 75K DNS records per second on average collectively. Each record in a DNS stream contains: *timestamp,..., [name; rtype; ttl; answer] <0,n>*
- **Netflow streams**: A set of Netflow records captured at the network ingress interfaces. For load-balancing purposes, the data is already divided into 26 different streams. These streams input 1M Netflow records per second on average. Each record in a Netflow stream contains: *..., srcIP, dstIP, ..., timestamp, packet, bytes*

We also deploy our system on a smaller European ISP with one DNS stream carrying 115K DNS records per second and two Netflow streams with 138K Netflow records per second.

Each of the above-mentioned streams has an internal buffer to be used in case the reading speed is less than their actual rate. If that buffer overflows, the streams start to drop data. Throughout this paper, wherever we mention *loss on the streams*, we mean that the buffers are overflown and start to drop. Therefore, the goal is to keep the buffer usage stable to avoid any loss.

Reading from multiple streams requiring shared memory access, and keeping the DNS records in memory to be quickly accessible makes this correlation challenging in terms of memory and CPU usage. To overcome these challenges, we leverage different techniques, details of which are explained in Section 3. In accordance with the data provider agreement, we refrain from reporting the exact values of the traffic, and all the traffic volume data throughout the paper is normalized.

3 METHODOLOGY

The goal is to categorize source of the traffic by their service in near real-time, e.g. to understand what fraction of the traffic is originated from Netflix, Amazon Prime, Google, etc. To realize this, we look for the IP address of the Netflow records in the *answer* section of the A/AAAA DNS records to find the *name* it corresponds to. Then looking at the CNAME records, we search for that *name* to find the corresponding CNAME. The results from this correlation are then correlated with BGP information to find the information such as source/destination ASes for each service. For the sake of brevity, in this work, we only focus on DNS-Netflow correlation. We note that the system is not bound to NetFlow data and can be adapted to use other data formats containing IP addresses and timestamps in a configuration file.

3.1 Overview

To perform the DNS-Netflow correlation, as Figure 1 shows, the DNS streams are received by *FillUp* workers. Multiple FillUp workers are allocated to each DNS stream to enable parallel processing of each shard of the DNS stream. These workers analyze the DNS records and fill up a shared internal storage with the DNS records. At the same time, the Netflow streams are received by the *LookUp* workers. These workers look for the source of the traffic in the shared internal storage. Again, we assign multiple LookUp workers to each Netflow stream. In our work, we are interested in analyzing the source of the traffic, hence we use the source IP address. Nonetheless, destination address or both source and destination addresses can be used with minor modifications. Then the result of this lookup is written onto the disk by the *Write* workers. Each worker has an input and output queue which enables the communication between workers. It is important to avoid that too many workers write to the same queue, as this contention causes a decrease in performance. Since multiple instances of the LookUp workers will try to simultaneously access a shared data structure where DNS records are kept, we split the DNS data and distribute them to different splits to then isolate each split as much as possible. In Section 4, we discuss whether this further splitting is needed. In Section 3.2 and Section 3.3, we go through the steps starting from reading the streams, to fill up the internal storage, and then look up and write.

We cannot strictly apply DNS records TTL and expire them after the TTL has passed, since there are multiple levels of caching and each might apply a different tolerance for expiring the records. Moreover, applying the actual TTLs on the DNS records requires regular iterations over all DNS data to check their TTL expiration. This degrades the performance dramatically and increases the memory usage (cf. Appendix A.8). On the other hand, we cannot keep the DNS records forever due to memory constraints. Therefore, we need to clear up the storage. We observe that 99% of the A/AAAA

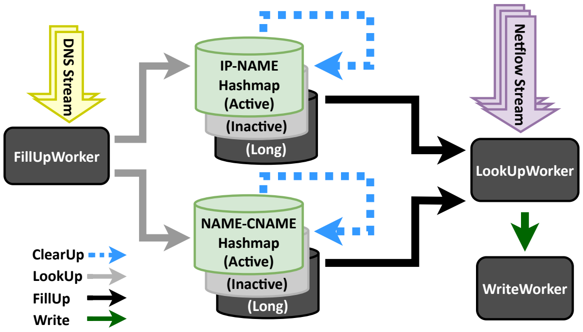


Figure 1: FlowDNS correlation architecture.

and CNAME records have a TTL smaller than 3600 and 7200 seconds, respectively (Ref. Appendix A.6). Therefore, we clear up the A/AAAA records every 3600 seconds, and CNAME records every 7200 seconds. However, since clearing the whole storage will remove all the states, we perform buffer rotation before clear-up. The internal storage where we keep the DNS records is a hashmap with the DNS *answer* section as key, and the *query name* as value. For implementing these hashmaps, we use the *concurrent-map* module in Go [18], which allows for high-performance concurrent reads and writes by sharding the map.

We add the DNS records in a primary hashmap, i.e., the *active* hashmap. After a certain amount of time has passed, we copy the contents of the active hashmaps into a secondary storage, i.e., the *inactive* hashmap, and clear up the active hashmap. In the next clear-up round, the current contents of the inactive hashmap will be overwritten by the new contents. Active hashmaps are actively updated with the newly arrived DNS records, while inactive hashmaps are only updated when the active hashmaps are cleared. A very small fraction of the DNS records has TTLs longer than the clear-up interval. Therefore, in case a DNS record's TTL is larger than a certain threshold, we put it into specific hashmaps which are never cleared or are cleared much less frequently, namely, long hashmaps. Otherwise, it is stored in the active hashmap. From now on, we call the active/inactive hashmaps for A/AAAA and CNAME records IP-NAME$_{\text{active/inactive}}$, and NAME-CNAME$_{\text{active/inactive}}$ respectively. See Appendix A.1 for an overview of the parameters and hashmaps used in FlowDNS.

3.2 DNS Processing

This part of FlowDNS takes in DNS streams and fills up an internal shared storage with the DNS records. These records will then be accessed in the Netflow Processing.

(1) DNS Streams are received by separate threads.
(2) The DNS records go through a filter to check if they are valid DNS responses.
(3) Valid DNS responses are added to a queue, namely FillUp Queue, to be then processed each by several FillUp workers. We need this queue to facilitate the synchronous execution of different workers.
(4) Each FillUp worker picks a DNS response from the FillUp Queue and if it is an A/AAAA record, labels it based on the IP address. This label will be used as a hashmap index later on.
(5) The FillUp worker then puts the DNS response in the shared hashmaps. In all our hashmaps, the key is the answer section, and the value is the query. We leverage two kinds of hashmaps:

- **IP-NAME hashmap:** Maps the answer section in an A/AAAA response, i.e. the IP address, to the queried domain name. We divide these hashmaps into several splits. We empirically find that 10 splits are suitable for our scenario. This can change for any deployment depending on the traffic volume. If, in Step 4, the IP for an A/AAAA response gets the label n, $0 \leq n < 10$, it goes to *IP-NAME*$_n$.
- **NAME-CNAME hashmap:** Maps the answer section, i.e. the domain name, to the queried canonical name for CNAME records.

(6) The FillUp worker keeps track of the timestamp in each DNS record. If *AClearUpInterval* seconds has passed, it copies the contents of *IP-NAME*$_{active}$ to *IP-NAME*$_{inactive}$ and clears *IP-NAME*$_{active}$. If *CClearUpInterval* seconds is passed, it copies the contents of *NAME-CNAME*$_{active}$ to *NAME-CNAME*$_{inactive}$ and clears *NAME-CNAME*$_{active}$.

See Appendix A.2 for the pseudocode of the above algorithm.

3.3 Netflow Processing

In parallel with the DNS processing, this part of FlowDNS takes in the Netflow streams, takes the source IP address, and looks for this IP address in the internal shared storage to find the corresponding domain name. It then writes the results into the output files.

(1) Netflow Streams are received by separate threads.
(2) The Netflow records go through a filter to check if they are valid Netflow records.
(3) The valid Netflow records are added to the LookUp Queue to be processed each by several LookUp workers.
(4) Each LookUp worker picks a Netflow record from the LookUp Queue and labels it based on the *srcIP* field.
(5) The LookUp worker looks for the *srcIP* in the *IP-NAME*$_{active\ n}$ hashmap, if the label from Step Item 4 is n. If nothing is found, it looks into the Inactive hashmap, and next into the long hashmap. If a *Name* is found, the search continues onto the next step. Otherwise, the search finishes here for that *srcIP* (*result* = NULL).
(6) The search will be continued in *NAME-CNAME*$_{active\ n}$ to find the CNAME for that *Name*. If a *CName* is found, the search continues to the next step. Otherwise, the search continues in the Inactive and then the long hashmap. If nothing is found, the search finishes here for that *Name* (*result* = *Name*).
(7) The search in the NAME-CNAME map continues until no further CNAME is found or a pre-defined loop limit is reached (*result* = *CName*). Our experiments show that a loop limit of 6 is sufficient for more than 99% of the records (ref. Appendix A.4). If the *result* is found with more than one look-up in NAME-CNAME maps, we add it to *NAME-CNAME*$_{active}$ for later use.
(8) The *result* along with the original Netflow is then passed to the Write Queue to be written in the output file by WriteWorkers.

See Appendix A.3 for the pseudocode. We publish the code for FlowDNS [14] for future researchers or network operators.

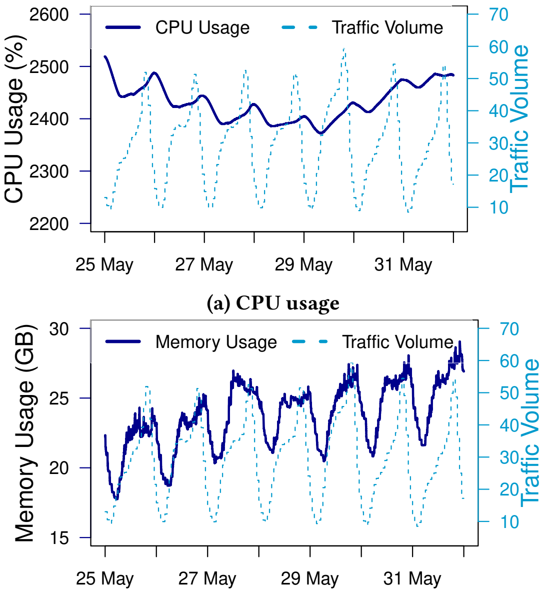
4 EVALUATION

In this section, we evaluate the matching accuracy and performance metrics for FlowDNS implemented in Go. First, we analyze the final version of FlowDNS on a full week of traffic of a large European ISP. Second, we selectively remove implementation features from FlowDNS on a one-day traffic capture to understand their importance by evaluating the effect on matching accuracy, CPU usage, and memory consumption. We evaluate all the benchmarks on an Ubuntu 18.04.5 LTS machine with 128 cores and 756 GB RAM. Figure 2 shows the CPU and memory usage of FlowDNS over one week when deployed at a large European ISP. In both plots, the right axis shows the traffic volume to compare the CPU/memory usage patterns with the traffic load. For all three metrics—traffic volume, memory usage, and CPU usage—we can clearly identify diurnal patterns, with daily peaks in the evening period, a low time during night hours, and an increase during the day. Note that we normalize the traffic volume in the right Y-axis. We show CPU usage as percentages, i.e., every 100% means 1 fully utilized CPU core. The CPU usage is around 2500% which means roughly 25 CPUs are used. Memory usage also oscillates between 15 GB and 30 GB. In addition to the large European ISP, we also deploy FlowDNS on a smaller network. On the smaller network, we observe average memory usage of 6 GB, and CPU usage of around 300%, both following a diurnal pattern. The ratio of CPU usage and number of flows remains the same in both deployments. The memory usage, however, is affected by both number of DNS records and number of parallel threads. This results in lower memory usage in the smaller network. On both deployments, the results are written to disk by a maximum delay of 45 seconds, and without any significant loss, i.e. 0.01% loss, on the data stream buffers. The ratio of correlated traffic to the total traffic, i.e. the correlation rate, is 81.7% on average for both deployments. We cannot correlate 18.3% of the traffic since (1) the coverage of our DNS data is only 95%, as discussed later in this section, and (2) not all the traffic is DNS-related, i.e. not all traffic has the destination IP address obtained through a DNS query.

Now, we remove the techniques used in the fully featured version of FlowDNS once at a time, introducing four new benchmarks:

- *No Split*: The hashmaps are not divided into several splits.
- *No Clear-Up*: The hashmaps are kept in memory forever.
- *No Rotation*: The hashmaps are cleared, but no buffer rotation takes place and no Inactive hashmap exists.
- *No Long Hashmaps*: The hashmaps are cleared up, and buffer rotation takes place, but records with large TTLs are also written in the Active hashmaps, instead of the Long hashmaps.

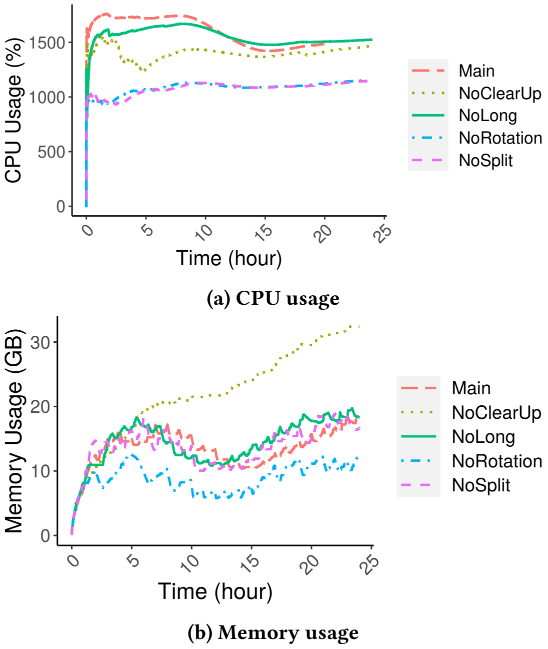
Figure 3 shows CPU and Memory usage for the above benchmarks. As expected, memory usage for *No Clear-Up* grows steadily over the day and can easily hit the memory limit. The mean correlation rate for this benchmark is 82.8%. The *No Rotation* benchmark uses much less memory compared to other benchmarks since it does not keep a copy of the original contents before clear-up. However, the average correlation rate for this benchmark is 79.5%. The *No Long Hashmaps* save neither a significant amount of memory nor CPU, yet, with a correlation rate of 81.1%, reduce the correlation rate by 0.6% compared to the Main benchmark. Therefore, the Long Hashmaps help keep those DNS records from being cleared up without much cost. The *No Split* neither improves nor degrades the memory usage but decreases the CPU usage significantly. This could be due to the reduced effort to access separate hashmaps simultaneously. The average correlation rate is also 81.7%. However, this should not be interpreted as if sharding the data is not helpful at all. In contrast, as we have used data sharding already both in our hashmaps, and in our job queues, explained in Section 3.1. The fact that this feature does not help as much as the others only shows



(a) CPU usage

(b) Memory usage

Figure 2: CPU and memory usage for *Main* benchmark over a week.



(a) CPU usage

(b) Memory usage

Figure 3: CPU and memory usage for different variants over a day.

that no further splitting is needed in our case. See Appendix A.5 for correlation rates per hour for all the benchmarks.

As we have seen with these four benchmarks, all implemented features in FlowDNS, except for IP-splitting, help increase the correlation rate while keeping the CPU and memory usage low.

Coverage. FlowDNS receives DNS cache misses gathered from the clients' default ISP resolvers. This data is sent from the ISP resolvers to our collectors via TCP. Therefore, even if the client uses DNS encryption while still using the default ISP resolver, the results from FlowDNS are not affected. However, if the clients use a resolver other than the default ISP resolver, e.g. a public DNS resolver (e.g., Cloudflare's 1.1.1.1, Google Public DNS, or Quad9), the DNS record is not received and therefore, FlowDNS can not correlate the Netflow traffic for those clients. To understand the number of DNS records we lose due to clients using public DNS resolvers, we analyze a sample 1-hour Netflow data and filter DNS and DoT traffic, i.e. ports 53 and 853. Then, using a public DNS resolvers list [11] and comparing it with our sample, we observe that 1 out of every 20 DNS packets is sent to a public DNS resolver. Therefore, the coverage of our DNS data is 95%.

Accuracy. Earlier in Section 4, we report that the correlation rate for FlowDNS, i.e. the number of bytes that could be correlated with *any* service/domain name compared to the total traffic volume in bytes, is 81.7% on average. However, this metric does not show whether the correlated service is the service actually used by the clients. Since we cannot access the actual domain names used by the clients, there is no ground truth against which we can compare our results. Nevertheless, we can estimate the accuracy of FlowDNS, by pinpointing the scenarios that could result in an incorrect service correlation and estimate their impact on the system's accuracy.

In FlowDNS, we keep the DNS data in a hashmap with the IP address as the key and domain names as values. Therefore, by design, observing multiple IP addresses for one domain name in the DNS data does not affect the accuracy of FlowDNS. However, observing multiple domain names for one IP address can affect the accuracy. In case a second domain name is observed with the same IP, i.e. the same key, the existing (first) domain name is overwritten by the second domain name, which in turn decreases the accuracy of our system. To confirm this, we design a small-scale accuracy analysis using generated traffic data. We browse two different websites and capture the traffic. Then, we extract the DNS packets from the captured traffic and feed them to FlowDNS as the DNS stream. We then create Netflow records from all traffic packets and feed them to FlowDNS as the Netflow stream. Finally, observing the correlated domain names and comparing them to the actual scenario, we find whether the system has correlated correctly. We consider two scenarios for this experiment: (1) Two websites with different domain names and different IP addresses. (2) Two websites with different domain names, using the same IP address. In the first scenario, we observe that all the traffic is correlated correctly, while in the second scenario, all the traffic is correlated to the second domain name. In other words, we had an accuracy of 100% and 50% in the first and second scenarios, respectively.

To estimate the impact of such mislabelling events, we analyze the domain name distribution per IP address. To this end, we analyze a 300-second sample of DNS records since as Figure 8 shows, more than 70% of the DNS records have TTL < 300 seconds. We observe that 88% of IP addresses are mapped to only a single domain name, as shown in Figure 9. We also did the analysis with a 1-hour sample and observed similar results. Therefore, we expect our results to be accurate for 88% of IP addresses in our flow data.

5 USE CASES

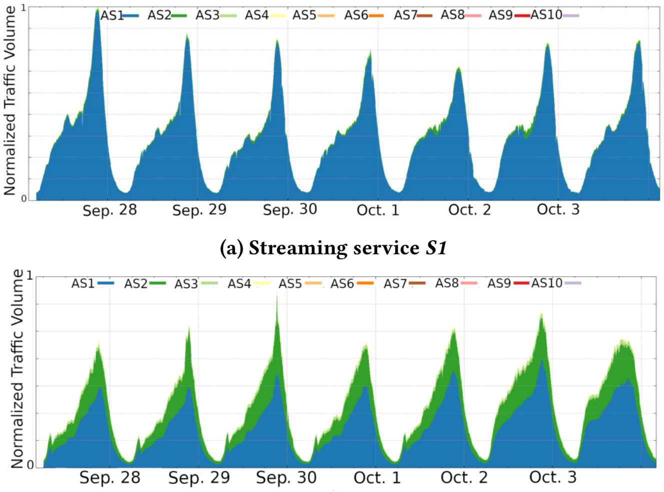
FlowDNS helps ISPs to better plan their networks, while providing the opportunity to analyze the traffic originated by malicious IDN homographs and spam domain names. There have been several studies on detecting malicious or unwanted domain names [1, 20, 26, 29], detecting IDN homographs [27, 30, 31], and also analyzing domain classification services [28]. However, to the best of our knowledge, there is no work measuring the traffic going to/originated by these domains. In this section, we illustrate three example use cases of FlowDNS, measuring the traffic from malicious or malformed domain names.

For all the following use cases, we use the correlated traffic for over a day in a large European ISP, including 39M unique domain names, and analyze the traffic originated by these domain names.

Network Provisioning and Planning. FlowDNS is already deployed in a large European ISP and a smaller European ISP. The output from FlowDNS is then correlated with BGP data, e.g. source AS, destination AS, hand-over AS, etc., to gain more knowledge about the path the traffic of a specific service takes. Figure 4 shows the contribution of different source ASes to the traffic volume of streaming services S1 and S2 over a week at the ISP. As Figure 4a shows, the traffic corresponding to the streaming service S1 is originated mostly from only one AS, while the streaming service S2 is originated mainly by two ASes as shown in Figure 4b. Note that AS numbers in two figures do not represent the same ASes necessarily. In Figure 4 we observe a diurnal pattern with slight differences between the two services. Knowing the source and intermediate ASes serving a specific service helps ISPs to negotiate with content providers over using ISP's resources instead of a third-party CDN. Also, in case of a broken peering link, it helps find the fallback paths, if they will be overloaded, and which services are effected.

Spam Domains. Using our 1-day traffic capture, we check the correlated domain names with the Spamhaus DBL (Domain Block List) [25] to see if any spamming, phishing or otherwise suspicious domains are generating any traffic. To avoid bandwidth limitations on Spamhaus DBL, we sample all the domain names once every hour, giving ca. 1M domain names, out of which 612 are classified as suspicious by the Spamhaus DBL. These include 512 *spam/generic bad reputation domains*, 41 *botnet C&C domains*, 34 *abused spammed redirector domains*, 11 *malware domains*, and 3 *phishing domains*. Collectively, these suspicious domain names originate multiple terabytes of traffic. Figure 5 shows a cumulative distribution of traffic volume per number of domain names for each of the above categories. In other words, it shows how many domain names contribute to what fraction of the traffic volume. As can be seen, a significant amount of traffic comes from spam and botnet domains, while only a limited number of domain names account for a large fraction of the traffic.

Malicious websites usually change their domain names rapidly to avoid being detected. Therefore, spam detection datasets such as Spamhaus DBL have an expiry date for their labels, i.e. if checked



(a) Streaming service *S1*

(b) Streaming service *S2*

Figure 4: Cumulative traffic volume for streaming service *S1* and *S2* per source AS.

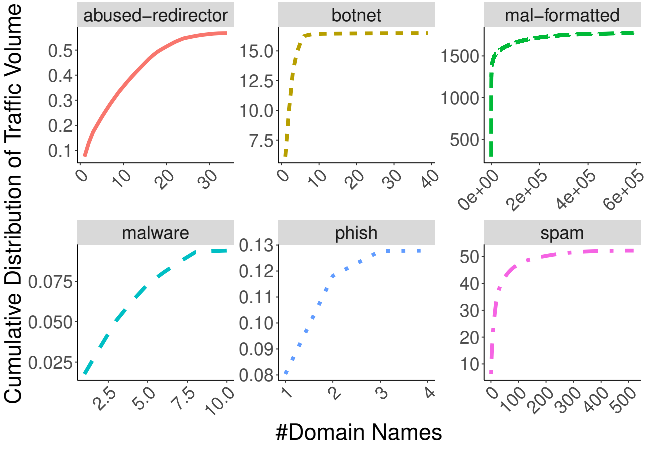


Figure 5: Cumulative distribution of the traffic volume per number of domain names.

after the expiry date, they will no longer exist in the dataset and therefore be labeled as benign. FlowDNS allows for real-time checking of the domain names with such datasets.

Invalid Domain Names. RFC 1035 stipulates specifications of DNS domain names [16]. In the following analysis, we focus on three rules to which valid domain names must adhere:

- The total length of the domain name is 255 bytes or less.
- Each label in the domain name is limited to 63 bytes.
- Each label starts with a letter, ends with a letter or digit, and the interior characters are limited to letters, digits, and hyphens.

The word *label* refers to each part of the domain name separated by dots, i.e. if the domain is *A.B.C.com*, labels are: *A*,*B*,*C*, and *com*. In our 1-day traffic capture, we observe that 666k domain names violate at least one of the above-mentioned rules. Figure 5 shows that almost all the traffic comes from a very limited number of domain names, and the amount of traffic originated by such domain names is quite significant. Note that the traffic volume is normalized. The most common conflict with the above-mentioned rules is disallowed interior characters. The most common disallowed character found in 87% of the malformatted domains is the underscore character, i.e. "_". Finally, we investigate the overlap of invalid domain names with domains in the spam category and find that only four malformatted domains also appear in the spam category. To understand how clients treat these malformed domains, we investigate whether traffic is being exchanged for these domains. We observe that 2.7% of the clients which receive traffic from malformed domains, send traffic back to 23.6% of these malformed domains. This bi-directional traffic accounts for 1.9% of the packets, mostly related to non-web port numbers, e.g. OpenVPN and Kerberos. All other packets are originated by malformed domains and receive no answer.

6 LESSONS LEARNED

During the design of FlowDNS, we learned the following lessons:

- When following the CNAME chain, we had to limit the chain length to 6 due to performance reasons. In our experiments, we observed that less than 1% of CNAME chains are longer than 6.
- Splitting the data into several shards allows for higher parallelism, while consuming higher CPU for the same amount of data. Therefore, it is important to keep an eye on this trade-off.
- Buffer rotation, i.e. copying the data once before clearing it, helps to increase correlation percentage without substantial CPU or memory usage in the long run. Therefore, it provides a good trade-off between resource utilization and correlation percentage.
- Expiring DNS records using their exact TTLs induces an unnecessary contention over the shared memory making the loss rate reach over 90%. Using rotating buffers with a common expiry time instead of the exact value helps in gaining the same correlation rate compared to keeping the DNS records forever, with no loss and is much more resource-efficient.

We hope that these lessons will prove useful for fellow network application developers and researchers alike.

7 CONCLUSION

Inferring the services behind a certain traffic flow is not possible merely by looking at the IP addresses due to the prevalent deployment of CDNs. In this work, we presented FlowDNS, a system to correlate DNS and Netflow streams in real-time. We used several techniques such as splitting the data, rotating buffers, and specific hashmaps to keep track of longer-living DNS records. We evaluated each of these techniques and confirmed the usefulness of each. Then, using FlowDNS, we analyzed the domain names with known datasets to detect malicious domains and observed that a substantial amount of traffic is originated by these domain names. Moreover, we checked the adherence of those domain names to standardization rules and observed that 1.7% of all the domain names violate them. The traffic originated by such domains accounts for 0.5% of the daily traffic. Finally, we plan to make FlowDNS available to fellow researchers and network operators.

REFERENCES

[1] Sara Afzal, Muhammad Asim, Abdul Rehman Javed, Mirza Omer Beg, and Thar Baker. 2021. Urldeepdetect: A deep learning approach for detecting malicious urls using semantic vector models. *Journal of Network and Systems Management* 29, 3 (2021), 1–27.

[2] Paul Aitken, Benoît Claise, and Brian Trammell. 2013. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011. https://doi.org/10.17487/RFC7011

[3] Ali AlSabeh, Elie Kfoury, Jorge Crichigno, and Elias Bou-Harb. 2022. P4DDPI: Securing P4-Programmable Data Plane Networks via DNS Deep Packet Inspection. In *Proceedings of the 2022 Network and Distributed System Security (NDSS) Symposium*. 1–7.

[4] Zhouyu Bao, Wenbo Wang, and Yuqing Lan. 2019. Using Passive DNS to Detect Malicious Domain Name. In *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing* (Vancouver, BC, Canada) *(ICVISP 2019)*. Association for Computing Machinery, New York, NY, USA, Article 85, 8 pages. https://doi.org/10.1145/3387168.3387236

[5] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. 2011. Exposure: Finding malicious domains using passive DNS analysis.. In *The Network and Distributed System Security (NDSS) Symposium*. 1–17.

[6] Xunxun Chen, Gaochao Li, Yongzheng Zhang, Xiao Wu, and Changbo Tian. 2019. A Deep Learning Based Fast-Flux and CDN Domain Names Recognition Method. In *Proceedings of the 2019 2nd International Conference on Information Science and Systems* (Tokyo, Japan) *(ICISS 2019)*. Association for Computing Machinery, New York, NY, USA, 54–59. https://doi.org/10.1145/3322645.3322679

[7] Cisco. 2021. Cisco IOS NetFlow. https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html.

[8] B. Claise (Ed.). 2004. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational). https://doi.org/10.17487/RFC3954

[9] Sean Donovan and Nick Feamster. 2014. Intentional network monitoring: Finding the needle without capturing the haystack. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*. 1–7.

[10] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. 2017. Measuring {HTTPS} adoption on the web. In *26th USENIX Security Symposium (USENIX Security 17)*. 1323–1338.

[11] Digineo GmbH. 2022. *Public DNS Server List*. https://public-dns.info/ Accessed: 2022-10-05.

[12] Jason Kim, Hyojoon Kim, and Jennifer Rexford. 2021. Analyzing traffic by domain name in the data plane. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*. 1–12.

[13] Hailing Li, Longtao He, Hui Zhang, Kai Zhang, Xiaoqian Li, and Chenghai He. 2020. CDN-Hosted Domain Detection with Supervised Machine Learning through DNS Records. In *Proceedings of the 2020 The 3rd International Conference on Information Science and System* (Cambridge, United Kingdom) *(ICISS 2020)*. Association for Computing Machinery, New York, NY, USA, 144–149. https://doi.org/10.1145/3388176.3388206

[14] A. Maghsoudlou. 2022. *FlowDNS: Correlating Netflow and DNS Streams at Scale*. https://github.com/maganiss/FlowDNS Accessed: 2022-10-24.

[15] P. Mockapetris. 1987. *Domain names - concepts and facilities*. RFC 1034. RFC Editor. 1–55 pages. https://www.ietf.org/rfc/rfc1034.txt

[16] P. Mockapetris. 1987. *Domain names - implementation and specification*. RFC 1035. RFC Editor. 1–55 pages. https://www.ietf.org/rfc/rfc1035.txt

[17] Giovane C. M. Moura, John Heidemann, Ricardo de O. Schmidt, and Wes Hardaker. 2019. Cache Me If You Can: Effects of DNS Time-to-Live. In *Proceedings of the Internet Measurement Conference* (Amsterdam, Netherlands) *(IMC ’19)*. Association for Computing Machinery, New York, NY, USA, 101–115. https://doi.org/10.1145/3355369.3355568

[18] Orcaman. 2022. *A Thread-Safe Concurrent Map for Go*. https://github.com/orcaman/concurrent-map Accessed: 2022-10-16.

[19] Ramakrishna Padmanabhan, John P Rula, Philipp Richter, Stephen D Strowes, and Alberto Dainotti. 2020. DynamIPs: analyzing address assignment practices in IPv4 and IPv6. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*. 55–70.

[20] Gopinath Palaniappan, Sangeetha S, Balaji Rajendran, Sanjay, Shubham Goyal, and Bindhumadhava B S. 2020. Malicious Domain Detection Using Machine Learning On Domain Name Features, Host-Based Features and Web-Based Features. *Procedia Computer Science* 171 (2020), 654–661. https://doi.org/10.1016/j.procs.2020.04.071 Third International Conference on Computing and Network Communications (CoCoNet’19).

[21] Roberto Perdisci, Thomas Papastergiou, Omar Alrawi, and Manos Antonakakis. 2020. Iotfinder: Efficient large-scale identification of iot devices via passive dns traffic analysis. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE Computer Society, 474–489.

[22] Sivaramakrishnan Ramanathan, Anushah Hossain, Jelena Mirkovic, Minlan Yu, and Sadia Afroz. 2020. Quantifying the Impact of Blocklisting in the Age of Address Reuse. In *Proceedings of the ACM Internet Measurement Conference* (Virtual Event, USA) *(IMC ’20)*. Association for Computing Machinery, New York, NY, USA, 360–369. https://doi.org/10.1145/3419394.3423657

[23] Audrey Randall, Enze Liu, Gautam Akiwate, Ramakrishna Padmanabhan, Geoffrey M Voelker, Stefan Savage, and Aaron Schulman. 2020. Trufflehunter: cache snooping rare domains at large public DNS resolvers. In *Proceedings of the ACM Internet Measurement Conference*. 50–64.

[24] Ernie Regalado. 2014. *The Multi-CDN Strategy*. https://www.bizety.com/2014/05/09/multi-cdn-strategy/ Accessed: 2022-06-27.

[25] Spamhaus Project. 2022. Spamhaus DBL. https://www.spamhaus.org/dbl/.

[26] Xiaoqing Sun, Mingkai Tong, Jiahai Yang, Liu Xinran, and Liu Heng. 2019. HinDom: A Robust Malicious Domain Detection System based on Heterogeneous Information Network with Transductive Classification. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. USENIX Association, Chaoyang District, Beijing, 399–412. https://www.usenix.org/conference/raid2019/presentation/sun

[27] Hiroaki Suzuki, Daiki Chiba, Yoshiro Yoneya, Tatsuya Mori, and Shigeki Goto. 2019. ShamFinder: An Automated Framework for Detecting IDN Homographs. In *Proceedings of the Internet Measurement Conference* (Amsterdam, Netherlands) *(IMC ’19)*. Association for Computing Machinery, New York, NY, USA, 449–462. https://doi.org/10.1145/3355369.3355587

[28] Pelayo Vallina, Victor Le Pochat, Álvaro Feal, Marius Paraschiv, Julien Gamba, Tim Burke, Oliver Hohlfeld, Juan Tapiador, and Narseo Vallina-Rodriguez. 2020. Mis-Shapes, Mistakes, Misfits: An Analysis of Domain Classification Services. In *Proceedings of the ACM Internet Measurement Conference* (Virtual Event, USA) *(IMC ’20)*. Association for Computing Machinery, New York, NY, USA, 598–618. https://doi.org/10.1145/3419394.3423660

[29] Sandeep Yadav, Ashwath Kumar Krishna Reddy, A.L. Narasimha Reddy, and Supranamaya Ranjan. 2010. Detecting Algorithmically Generated Malicious Domain Names. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement* (Melbourne, Australia) *(IMC ’10)*. Association for Computing Machinery, New York, NY, USA, 48–61. https://doi.org/10.1145/1879141.1879148

[30] Ramin Yazdani, Olivier van der Toorn, and Anna Sperotto. 2020. A Case of Identity: Detection of Suspicious IDN Homograph Domains Using Active DNS Measurements. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroSPW)*. 559–564. https://doi.org/10.1109/EuroSPW51379.2020.00082

[31] Zicong Zhu, Tran Phuong Thao, Hoang-Quoc Nguyen-Son, Rie Shigetomi Yamaguchi, and Toshiyuki Nakata. 2020. Enhancing A New Classification for IDN Homograph Attack Detection. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing*. 507–514. https://doi.org/10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00093

A APPENDIX

A.1 Parameters and In-memory Storage

Table 1 shows an overview of the parameters and names of the in-memory storage that we use in FlowDNS. Note that $0 \leqq n < NUM\_SPLIT$.

A.2 DNS Processing Pseudocode

Algorithm 1 shows an overview of the fillUpWorker thread which we mentioned in Section 3.2. This function takes the DNS records and fills in the hashmap.

A.3 Netflow Processing Pseudocode

Algorithm 2 shows an overview of the lookUpWorker thread which reads the netflow records, looks them up in the Active, Inactive, and Long hashmaps and finds the results.

A.4 CNAME Chain Length Distribution

CNAME look-up can sometimes include multiple consequent look-ups with one CNAME mapping to another more than once. We studied the CNAME chain length, and as shown in Figure 6, we observed that more than 99% of the DNS records can be mapped with a chain of 6 look-ups. Therefore, we limit the number of CNAME chain look-ups to 6 in FlowDNS.

Parameter Name	**Description**
AClearUpInterval	Time in seconds after which the *IP-NAME* hashmap is cleared.
CClearUpInterval	Time in seconds after which the *NAME-CNAME* hashmap is cleared.
NUM_SPLIT	Number of splits for each IP-NAME hashmap.
Storage Name	**Description**
IP-NAME$_{\text{Active n}}$	Hashmap for DNS records with TTL < AClearUpInterval and label n.
IP-NAME$_{\text{Inactive n}}$	Hashmap where the contents of IP-NAME$_{\text{Active n}}$ are copied to every AClearUpInterval seconds
IP-NAME$_{\text{Long n}}$	Hashmap for the new DNS records with TTL >= AClearUpInterval and label n.
NAME-CNAME$_{\text{Active}}$	Hashmap for the new CNAME responses with TTL < CClearUpInterval.
NAME-CNAME$_{\text{Inactive}}$	Hashmap where the contents of NAME-CNAME$_{\text{Active}}$ are copied to every CClearUpInterval seconds
NAME-CNAME$_{\text{Long}}$	Hashmap for the new CNAME responses with TTL >= CClearUpInterval.

Table 1: Overview of Parameters and Storage Names.

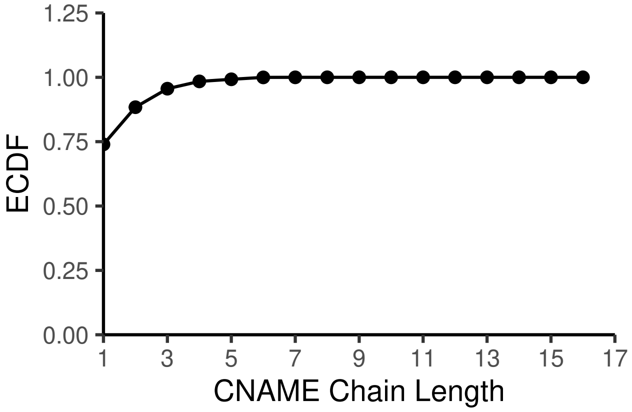


Figure 6: Cumulative distribution of CNAME chain length over a day.

A.5 Correlation Rate

The correlation rate, i.e. the ratio between correlated traffic and total traffic is illustrated in Figure 7 for different benchmark variants. The *No Split* benchmark excluded from the plot since it has a complete overlap with the *Main* benchmark. The top two variants in terms of correlation rate are *Main* and *NoClearUp*. The *NoClearUp* performs unacceptable in terms of memory usage. The lowest correlation rate belongs to *NoRotation*, which shows the importance of buffer rotation in FlowDNS.

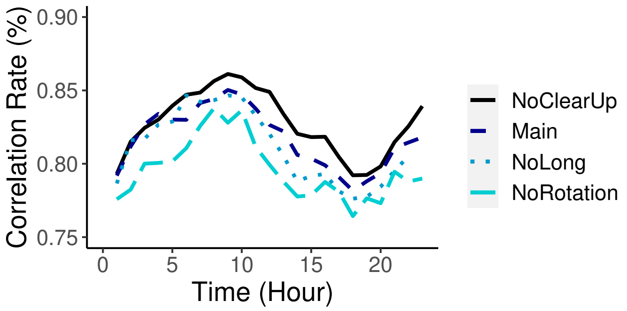


Figure 7: Correlation rate for benchmark variants.

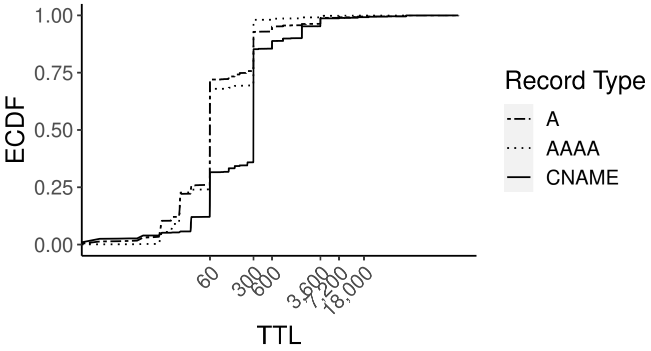


Figure 8: Cumulative distribution of TTLs for DNS records over a day.

A.6 DNS Records' TTLs

To find out the correct number for the clear-up intervals, namely *CClearUpInterval* and *AClearUpInterval*, we investigate the TTLs for the DNS records. We look at the DNS records' TTLs over a day at a large Europoean ISP, and find out that 99% of the A/AAAA and CNAME records have TTL smaller that 3600 and 7200 seconds respectively, shown in Figure 8. Therefore, in FlowDNS, we set the clear-up variables as follows:

$$CClearUpInterval = 7200$$

$$AClearUpInterval = 3600$$

A.7 Number of Domain Names per IP address

We look at a 300-second period of DNS records to investigate the number of domain names that map to the same IP address leading to a mislabelling event in FlowDNS. Figure 9 shows the cumulative distribution of number of domain names per IP address. We observe that 88% of the DNS records only map to one domain name in 300 seconds. Note that we choose 300 seconds since this is the TTL for 70% of our DNS records. We also analyze this in a 1-hour sample of DNS data and observe similar results. Additionally, we analyze the number of IP addresses per domain name in a 300-second period of DNS records. We observe that 35% of the domain names map to more than one IP address. We also analyze this in a 1-hour sample of DNS records and observe similar results. Note that observing multiple IP addresses per domain name, which is a significantly more probable event compared to multiple names for one IP address,does not effect the accuracy of FlowDNS.

Algorithm 1: DNS Read and Fill-up Overview

```
Function fillUpWorker DNSRecord d
    n = label(d);
    if d.rtype is A/AAAA then
        if d.ts - lastAClearUpTs >= 3600 then
            IpName.Inactive = IpName.Active;
            IpName.Active = {};
            lastAClearUpTs = d.ts;
        end
        if d.ttl <= 3600 then
            IpName.Active[n][d.answer] = d.query;
        else
            IpName.Long[n][d.answer] = d.query;
        end
    else
        if d.ts - lastCClearUpTs >= 7200 then
            NameCname.Inactive = NameCname.Active;
            NameCname.Active = {};
            lastCClearUpTs = d.ts;
        end
        if d.ttl <= 7200 then
            NameCname.Active[n][d.answer] = d.query;
        else
            NameCname.Long[n][d.answer] = d.query;
        end
end
```

Algorithm 2: Netflow Read and Look-up Overview

```
Function lookUpWorker NetflowRecord nf
    n = label(nf);
    Name = deepLookUp(nf.srcIP, IpNameObj[n]);
    loopCount = 0;
    if Name != Null then
        results = append(results, Name);
        Cname = deepLookUp(Name, NameCnameObj[n]);
        while Cname != Null and loopCount <= 6 do
            results = append(results, Cname);
            loopCount ++;
        end
    return results;

Def deepLookUp NetflowRecord nf, MapObj hm
    Name = NULL;
    if nf.srcIP in hm.Active then
        Name = hm.Active[nf.srcIP];
    else if nf.srcIP in hm.Inactive then
        Name = hm.Inactive[nf.srcIP];
    else if nf.srcIP in hm.Long then
        Name = hm.Long[nf.srcIP];
    return Name;
```

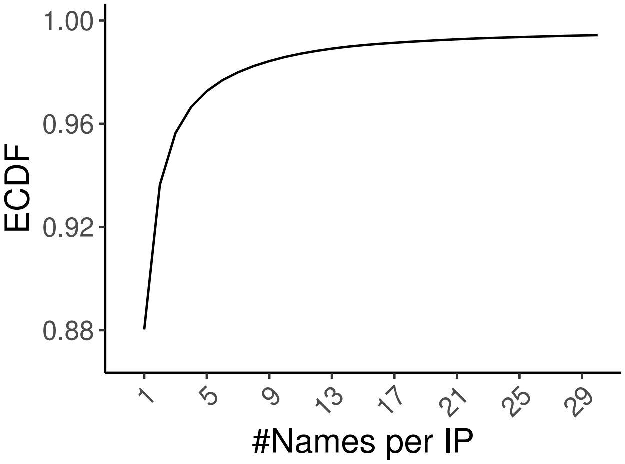


Figure 9: Cumulative distribution of number of domain names per IP address.

A.8 Applying the Exact TTLs

We try applying the exact TTLs from the DNS records on our correlation, meaning we correlate the IP from a DNS record with the source IP from the Netflow record only if the DNS record's TTL plus its timestamp is less than the timestamp from the Netflow record which we consider current timestamp. In other words:

$$TTL_{\text{dns}} + Timestamp_{\text{dns}} < Timestamp_{\text{netflow}}$$

We also run a regular process to clear-up the expired DNS records, when the above-mentioned condition does not hold. We run this on the same sources of data, meaning DNS and Netflow streams at the large European ISP. We observe that the internal buffers of all the streams start to overload from the very first minutes of running the above-mentioned system, with the loss rate of over 90% for both Netflow and DNS streams. We observe that the memory usage reaches up to 45 GB memory usage after only 1 hour of running the system. When we compare this to the results from FlowDNS in Figure 2b, we see that the memory usage is doubled although only 10% of the data is received at the system and others are lost. This could be due to the regular clear-up process not being fast enough to clear-up all the expired TTLs as the hashmaps grow, while at the same time, the contention to access the shared memory is so high that the performance degrades dramatically.