An Analysis of DepenDNS

Nadhem J. AlFardan and Kenneth G. Paterson*

Information Security Group (ISG)
Royal Holloway, University of London
Egham Hill, Egham, TW20 0EX, UK
emails: {nadhem.j.a.alfardan}{kenny.paterson}@rhul.ac.uk

Abstract. Recently, a new scheme to protect clients against DNS cache poisoning attacks was introduced. The scheme is referred to as DepenDNS and is intended to protect clients against such attacks while being secure, practical, efficient and conveniently deployable. In our paper we examine the security and the operational aspects of DepenDNS. We highlight a number of severe operational deficiencies that the scheme has failed to address. We show that cache poisoning and denial of service attacks are possible against the scheme. We also demonstrate a high factor amplification attack against DepenDNS, which can lead to a large scale Internet denial of service attack. Our findings and recommendations have been validated with real data collected over time.

Keywords DNS, DepenDNS, DNS cache poisoning, Denial of Service, Amplfication.

1 Introduction

The Domain Name System (DNS) [8] [9] is critical to the proper operation of the Internet. DNS provides the service of mapping names to IP addresses (for example, translating www.example.com to 192.0.32.10). This information is maintained on DNS servers in the form of persistent or cached entries referred to as Resource Records (RRs). DNS can be thought of as a distributed database with a hierarchal structure that is made of name servers hosting the database. The root domain (".") is at the top of the hierarchy and is served presently by 13 root servers that are distributed around the world [10]. The second level contains top-level domains (TLDs) that can be classified as generic (gTLD) such as .com, or country code (ccTLD) such as .uk. Multiple levels exist underneath the TLDs. The domain names located in the lower levels are generally served by their corresponding organisations. For example IANA would host the domain, "example.com" [2].

The operation of DNS is based on queries and responses. A client initiates the process by sending a resolution request for a host name (for example, www.example.com) to its DNS resolver which in return searches its cache entries for the host being requested. If an entry does not exist, the resolver may go through a recursive DNS look-up process. If the resolver does not have information about the TLD of the domain in the request, then the look-up process starts from the root servers and continues all the way down to the authoritative servers responsible for hosting the domain being requested (for example example.com). Information about domains and their records are contained within DNS zones. The authoritative servers maintain a DNS zone's database and reply to requests for hosts that exist in the zone. Upon receiving an answer from an authoritative server, a resolver will cache and forward the IP address to the requesting client.

DNS messages, including queries and responses, are communicated in clear using Unreliable Datagram Protocol (UDP) with no integrity check mechanisms in place [9]. This makes DNS vulnerable to attacks involving unauthorised data modification, in which an attacker may alter the data in various ways, with an ultimate objective of poisoning the content of a DNS resolver cache. Such attacks are referred to as DNS cache poisoning and they present potential security threats to users. A successful cache poisoning attempt can lead users to malicious websites for fraud or phishing purposes. To successfully poison the cache of a DNS server, an attacker may spoof a DNS reply and deliver it to the requester ahead of the legitimate one. Subsequent replies for the same DNS request are ignored by the requester. To be accepted by the requester, the spoofed reply must also pass the standard security controls incorporated within DNS such as comparing the DNS transaction identifier (TXID) and the randomised UDP source port in requests and replies [3]. Although each of these fields has 16 bit, some DNS resolver implementations contained flaws in the randomisation process causing insufficient entropy and hence a higher probability for an attacker to predict the values. This assumes that the attacker has no access to the DNS query and hence must guess the random variables. Other situations include the implementation of Network Address Translation (NAT) in front of DNS resolvers. A NAT device may replace the original random source port with a sequential one of its own, causing the same behaviour described earlier of reducing the unpredictability of the UDP source port. A more severe scenario is when the attacker acts as a Man-In-The-Middle

* This author's research supported by an EPSRC Leadership Fellowship, EP/H005455/1.

(MITM). In this case, the attacker has visibility of all DNS data and hence can reply to DNS queries using information of her choice.

Threats targeting DNS and its cache in particular are not new. They have existed since the day DNS was introduced. However, the topic gained significant visibility and attention after the publication of Kaminsky's DNS exploit in summer 2008. Kaminsky discovered a fundamental flaw within DNS implementations that could allow remote attackers to perform arbitrary cache poisoning within a matter of seconds [7].

Various security techniques have been proposed to protect against DNS cache poisoning attacks. These techniques can be implemented in the clients, resolvers, authoritative servers, or a combination of them. Examples of such techniques include Domain Name Cross Referencing (DoX) [15], 0x20-Bit Encoding [4], ConfiDNS [11] and DNS Security Extensions (DNSSEC) [1].

Unfortunately, the majority of the proposed DNS security techniques have not been adopted in practice. The main reason behind this is the significant effort required to change the underlying DNS infrastructure to accommodate these new techniques. A good example is DNSSEC which is one of the most visible initiatives to secure DNS. DNSSEC deploys cryptographic measures to ensure the authenticity of the DNS data exchanged by digitally signing the DNS records. However, the challenges associated with the practical implementation of DNSSEC have lead to a significant delay in deploying the technology [10].

A recently published DNS security technique is DepenDNS [13]. DepenDNS is proposed as a client-based DNS implementation designed to protect clients from cache posisoning attacks. The fundamental concept behind DepnDNS is sending the same DNS query to multiple resolvers and then evaluating the responses. The evaluation is based on an algorithm referred to as π in [13]. DepenDNS is supposed to be practical, efficient and secure according to [13].

The authors of [13] position DepenDNS as a comprehensive solution against cache poisoning attacks. Therefore, the scheme should be able to protect clients from various DNS cache poisoning attacks including the following three generic spoofing attack scenarios:

- **Scenario 1:** A spoofing attack against a client in which the attacker sends spoofed DNS replies to the client. We assume that the attacker has no access to the DNS requests and hence is not aware of the host names being requested. Let us suppose that this attack has a success probability of p_1 when DepenDNS is not deployed.
- **Scenario 2:** A spoofing attack against the DNS resolver. We assume that the attacker has no access to DNS requests and hence is not aware of the host names being requested. In this scenario, the attacker tries to poison the resolver's DNS cache for an arbitrary host name at any point in time. Let us assume that the attack has a success probability of p_2. The impact of this attack is higher compared to scenario 1 since it can affect all clients served by the targeted resolver when requesting entries that have been poisoned.
- **Scenario 3:** The attacker has control over the DNS resolver and hence has visibility of the DNS requests. The probability of success of a spoofing attack is 1 when DepenDNS is not deployed.

The reader can think of the above scenarios from an abstract point of view, in which the exact implementation is irrelevant. For example, a random 16 bit TXID may or may not be in use. The objective of using this approach is to evaluate the effectiveness of the scheme regardless of the underlying implementation. In addition, the spoofing approaches discussed above can also be used to conduct other types of attacks than cache poisoning, as we demonstrate in this paper.

When DepenDNS is deployed, clients should be able to detect and prevent the above three generic attacks and hence decrease their success probabilities to a minimal value. This is supposed to be achieved by queering multiple DNS resolvers and evaluating the responses against a set of pre-defined conditions.

Our Contribution

We analyse DepenDNS and highlight the scheme's shortcomings. We reveal some conditions under which we have been able to circumvent the scheme to perform cache poisoning, denial of service and amplification attacks. We have performed cache poisoning attacks while maintaining the same success probabilities of p_1, p_2 and 1 as described in scenarios 1, 2 and 3 respectively, making DepenDNS ineffective. In addition, we have executed a successful denial of service attack agains the scheme by forcing it to reject the IP addresses contained in all the DNS replies for the host name being requested. Although the objective of a denial of service attack is different from the goal of the three attack scenarios discussed in Section 1, we are able to demonstrate the following:

- The same approaches of spoofing responses can be used to achieve the denial of service goal, while maintaing the same success probabilities, making DepenDNS ineffective.
- The controls implemented by the scheme introduce a new category of attacks.

We also demonstrate how an attacker can turn an implementation of DepenDNS into a serious denial of service tool as a result of a simple amplification attack.

We focus on evaluating the security and deployability aspects of DepenDNS. Our approach consists of analysing the proposed scheme, investigating the existence of vulnerabilities and eventually attacking the scheme. **First**, we start with an explanation of the operation of DepenDNS and how algorithm π's calculations are performed. The reader will find that our explanation of algorithm π and the symbols we use differ slightly from the original DepenDNS paper [13]. Our aim is to give the reader a concise and clear description of the algorithm. **Second**, we provide a review of the scheme and highlight a number of unclear assumptions made in [13]. We also consider a number practical deployment issues that should have been addressed in [13]. **Third**, we analyse if DepenDNS is vulnerable to cache poisoning, Denial of Service (DoS) and amplification attacks. We have discovered scenarios in which we were able to successfully exploit the scheme. The attacks that we have performed against DepenDNS are based on a full implementation of the scheme and the use of real data collected over a period of time. We clearly highlight any assumptions made for our attacks to be successful. **Fourth**, we study the performance and accuracy of DepenDNS. We conclude the paper with a summary of our findings.

2 DepenDNS

DepenDNS has been recently proposed as a protection scheme against DNS cache poisoning attacks. The fundamental security objective of DepenDNS is to protect clients from bogus IP address sent by DNS resolvers. These bogus IP address would have either arrived from an already poisoned resolver's cache or as a result of a spoofed DNS reply message directed against the client. The former is the more likely scenario. An attacker would target a DNS resolver serving a large number of clients in order to achieve a higher impact. The scheme proposed in [13] describes how a client running DepenDNS can detect and reject such bogus IP addresses.

The scheme relies on forwarding the same DNS query to multiple resolvers and then evaluating the replies using an algorithm π. Algorithm π runs on the client's machine and accepts or rejects each IP address suggested by the resolvers. The decision is based on comparing a number of parameters against a set of pre-defined thresholds. Accepted IP addresses are passed to the client and are saved in a history table maintained by DepenDNS. The resolvers and authoritative servers are not involved in the IP addresses selection process done by DepenDNS.

The concept of invoking multiple resolvers in the DNS resolution process has been proposed by other DNS security schemes. For example, DoX [15] deploys a collaborative network of peers to protect against DNS cache poisoning. The motivation behind using multiple resolvers is that the probability of poisoning several resolvers at the same time should be much lower than that of poisoning one resolver. Although DepenDNS makes use of the same approach, the parameters defined and the calculations carried out by its algorithm π are different.

In this section we describe the decision making process used by DepenDNS. We also analyse the proposed scheme and highlight some unclear assumptions and unsupported claims made in [13].

2.1 DepenDNS Algorithm π

The decision making process of DepenDNS is carried out by algorithm π. This algorithm expects the following inputs:

- The IP addresses returned by all the resolvers being queried for the given host name.
- Access to the history table containing the previously accepted IP addresses for the given host name. This is a separate table that is maintained by DepenDNS and is different from the client's DNS cache table. In Section 2.2 we further analyse the suggestions made in [13] on how to build and maintain the history data.

The output of algorithm π is a set of IP addresses that are supposed to be legitimate for the host name being requested. The output is used to update the history table and is forwarded to the requesting entity on the client's machine. An example of a requesting entity is a web browser. It is important to highlight that DepenDNS does not maintain history information about *rejected* IP addresses. Algorithm π defines the following parameters:

- t is the number of the resolvers to which the client is configured to send its DNS request messages.
- R_j is the set of IP addresses returned by the j^{th} resolver, where $1 \leq j \leq t$. We write $R_j = \{IP_{1_j}, IP_{2_j}, ..., IP_{l_j}\}$ where l_j is the number of IP addresses returned by the j^{th} resolver. In practice, a DNS reply may contain duplicate IP addresses. DepenDNS normalises the reply by removing the repeated IP addresses and including a single copy of each IP address in R_j.
- R is the set that contains all the distinct IP addresses in the replies from t resolvers, i.e. $R = R_1 \cup R_2 \cup ... \cup R_t$. We write $R = \{IP_1, IP_2, ..., IP_m\}$ where m is the number of the distinct IP addresses returned by the t resolvers.
- n_j^i is a variable that is set to 1 if $IP_i \in R_j$ and 0 otherwise.
- n^i is the number of times IP_i appears across all R_j, i.e. $n^i = \sum_{j=1}^{t} n_i^j$.
- $n^{\max} = \max(n^1, n^2, ..., n^m)$.

- $c^k_{current}$ is a variable with value between 0 and 1. $c^k_{current}$ is calculated by dividing the number of occurrences of IP addresses in all R_j that share the same leftmost 16 bit (represented by the integer k) by the total number of IP addresses returned by the t resolvers. IP addresses that share the same leftmost 16 bit are considered to be part of the same class, k.
- H is the history data maintained by DepenDNS. H contains the IP addresses that have been accepted by algorithm π for each host name.
- $c^k_{history}$ is a variable parameter with value between 0 and 1. $c^k_{history}$ is calculated in a similar way as $c^k_{current}$ but uses the information in H for the host name being requested as input for calculation.
- A is the set of IP addresses that are accepted by algorithm π for a specific DNS request. Each run of algorithm π generates a new A.

In general, algorithm π makes the decision to accept or reject an IP address after examining data related to the number of occurrences, history information and the leftmost 16 bit of that IP address. For each address IP_i, algorithm π calculates the variables α_i, β_i, and γ_i as follows:

1. α_i can be thought of as an indicator for the distance between n^i and n^{max}. α_i is determined by comparing n^i to n^{max} along with a tolerance variable that is set to 20% in [13].
$$\alpha_i = \begin{cases} 1, & \text{if } n^i \geq (0.8 \cdot n^{max}); \\ 0, & \text{otherwise.} \end{cases}$$
2. β_i is related to the history data of DepenDNS. β_i is set to 1 if the IP address for the host name under evaluation exists in H. This indicates that the IP address has passed the evaluation process at some earlier point in time.
$$\beta_i = \begin{cases} 1, & \text{if } IP_i \text{ exists in the history data, } H; \\ 0, & \text{otherwise.} \end{cases}$$
3. γ_i is related to the leftmost 16 bit of the IP address and is determined by comparing $c^k_{current}$ and $c^k_{history}$. γ_i is set to 1 if the absolute difference between $c^k_{current}$ and $c^k_{history}$ is at most 0.1.
$$\gamma_i = \begin{cases} 1, & \text{if } IP_i \text{ belongs to } k^{th} \text{ class and} \\ & -0.1 \leq c^k_{current} - c^k_{history} \leq 0.1; \\ 0, & \text{otherwise.} \end{cases}$$

Once α_i, β_i, and γ_i are calculated for each IP_i, algorithm π constructs the following sets:

- R_α which contains the IP addresses in R with $\alpha_i = 1$. $R_\alpha = \{IP_i \in R : \alpha_i = 1\}$
- R_β which contains the IP addresses in R with $\beta_i = 1$. $R_\beta = \{IP_i \in R : \beta_i = 1\}$
- R_γ which contains the IP addresses in R with $\gamma_i = 1$. $R_\gamma = \{IP_i \in R : \gamma_i = 1\}$

Algorithm π then calculates N, which is referred to as the dispersion strength in [13]. N is calculated as follows:

$$N = \frac{|R_\alpha \cup R_\beta \cup R_\gamma|}{\text{Mode}(|R_1|, |R_2|, ..., |R_t|)}$$

Upon calculating N, algorithm π proceeds to calculate the grade, G_i, for each address IP_i. The value of G_i determines whether an IP address is accepted or not. G_i is calculated as follows:

$$G_i = \alpha_i \cdot (G_\alpha - 10 \cdot (N-1)) + \frac{1}{2} \cdot (\beta_i + \gamma_i)(G_{\beta\gamma} + 10 \cdot (N-1)),$$

where G_α and $G_{\beta\gamma}$ represent the weights given to α and $\beta\gamma$ and are set to 60 and 40 respectively in [13]. Note that N is the only variable in the above equation, since the values of G_α and $G_{\beta\gamma}$ are fixed. IP addresses with grades higher than or equal to 60 are accepted and are used to update A and H.

2.2 Scheme Review

According to [13], a good percentage of end-points should be able to make use of DepenDNS, since it is a client-based scheme. This may result in a large number of clients running DepenDNS. Such a potential large deployment of a scheme should not only consider the security aspects of the scheme but should also study the deployment challenges and the expected practical impacts.

In the previous section we described the calculations performed by algorithm π along with the decision making process. To reach a decision on whether to accept or reject an IP address, the scheme makes a number of explicit and implicit assumptions. Unfortunately, some of these assumptions in [13] have not been justified or backed by supporting information. In addition, the scheme has not addressed some important operational aspects of DNS. In this section we examine the validity of some of the assumptions made in [13]. We also highlight some of the characteristics of DNS that the proposal in [13] has failed to recognise.

System Initialisation: A client running DepenDNS needs to be configured with the IP addresses of the DNS resolvers that it needs to query. The method by which the resolvers' IP addresses are set on the client is not discussed in [13]. This might seems to be a minor issue but we believe that it has a great operational impact in the case of large scale deployments. Manual configuration of the resolvers' IP addresses is impractical and hence an automated IP assignment approach should be considered. Specifying the IP assignment method is out of this paper's scope, but a network protocol such as DHCP would be needed.

Managing the History Data of DepenDNS: Variables that are related to the history data maintained by DNS have a significant influence on the decisions made by the scheme. The values of β_i and γ_i are determined by the existence of history data and are part of the grade calculation. The values of β_i and γ_i are set to 0 in case no information exists in the history for the host name being requested. This clearly introduces an operational challenge since the history is expected to be empty initially. To address this challenge, [13] proposes either deploying a centralised database that can be used when new domains are queried, or adjusting the values of G_α and $G_{\beta\gamma}$ accordingly. The first option is unrealistic and cannot be practically deployed: there are more than 180 million domain names [5] and there exists no centralised database that contains information about all the domains; even if one existed, it would be impossible to maintain such a database and track changes in domains' information around the world. The second option can be implemented. However, the proposed changes in the values of G_α and $G_{\beta\gamma}$ are not included in [13] and there is no assurance provided that such changes will not negatively impact the security properties of the scheme.

Tolerance Value Used to Calculate α: The value of α_i is set to 1 if $n^i \geq (0.8 \cdot n^{max})$ for IP_i. Else, α_i is set to 0. The use of a 20% tolerance level is not justified, nor is any guideline on how to select a suitable value is given. We would have expected more detailed discussion of how to select such critical systems parameters.

Tolerance Value Used to Calculate γ: The value of γ_i is set to 1 if $-0.1 \leq c^k_{current} - c^k_{history} \leq 0.1$ for IP_i. Else, γ_i is set to 0. As we α, the use of a 10% tolerance level for γ should have been justified.

Class Consideration by γ: Algorithm π determines the value of γ_i based on the leftmost 16 bit of IP_i. The authors of [13] claim that a domain name may have several IP addresses but these IP addresses usually share the same leftmost 16 bit. However, no evidence or experimental data to support such a claim is offered in [13].

Number of Resolvers: The proposed implementation of the scheme considers the use of 20 resolvers. However, the proposal does not explain the reasons behind choosing this number of resolvers. We use this number when analysing the scheme's behaviour in the coming sections. On the other hand, enterprise networks generally deploy a small number of resolvers internally, typically 2 or 3. Adding 20 resolvers for the sake of implementing DepenDNS is clearly an expensive exercise. Although service providers deploy a larger number of servers compared to enterprises, only few might employ such a number of resolvers. This introduces another challenge, which is the method through which the DNS resolvers are selected.

2.3 DepenDNS and Content Delivery Networks

Content Delivery Networks (CDNs) are built to enhance the user's experience when trying to access an Internet resource like a website. A Content Delivery Network consists of a set of surrogate servers distributed around the world. The surrogate servers are deployed in multiple locations in order to optimise the end user experience by choosing the nearest surrogate server to the user [14]. For example, web requests generated by a UK-based end user for a website hosted by a CDN will generally be served by a surrogate server that is located in the UK. Most CDN providers deploy DNS redirection to forward the client's request to the closest server containing the resource being requested. One of the characteristics of DNS records served by CDNs is that they have a low Time To Live (TTL) value. By way of example, the following shows the TTL value for "134.g.akamai.net", which is the Canonical Name (CNAME) record corresponding to "www.live.com". It can be seen that the TTL value is set to only 20 seconds.

```
$ dig www.live.com
...
www.live.com. 1216 IN CNAME search.msn.com.edgesuite.net.
search.msn.com.edgesuite.net. 2382 IN CNAME a134.g.akamai.net.
a134.g.akamai.net. 20 IN A 88.221.94.72
a134.g.akamai.net. 20 IN A 88.221.94.34
...
```

CDNs have proven lately to be a very attractive option for hoting rich web content such as video. In fact, high profile websites such as YouTube, CNN and BBC are making use of commercial CDNs such as Akamai and Limelight [12].

Solutions which attempt to address security or performance challenges related to DNS must consider the characteristics of CDNs. In this paper we analyse the behaviour of DepenDNS when the domain name being requested is served by a CDN. We have been able to attack an implementation of DepenDNS when the host names being requested are hosted by CDNs.

3 Attacking DepenDNS

The implementation of DepenDNS is supposed to provide a good level of protection against DNS cache poisoning attacks. In Section 1 we referred to three attack scenarios that clients running DepenDNS should be able to detect and prevent. Each attack has its own probability of success. In this section, we explore how the scheme behaves under a number of conditions with the intention of trying to find and exploit vulnerabilities in the scheme. We were able to find conditions under which we can poison the cache of DepenDNS, perform a denial of service attack against the scheme and execute amplification attacks that can trigger the generation of high volume of network traffic. Our cache poisoning and DoS attacks show that implementing DepenDNS had no effect in lowering the probability of success of our three attack scenarios identified in Section 1. We state any assumptions we make for our attacks to be successful.

3.1 General Assumptions

Our general assumptions are as follows:

Assumption 1 *The attacker knows the IP address of one of the t resolvers that the client communicates with.*

Assumption 2 *We bound the attack to a single resolver. This assumption is made to make our attack model realistic and also considers a worst case scenario for the attacker: If an attack against DepenDNS is successful when a message from a single resolver is bogus, then it will certainly be successful when two or more resolvers are targeted.*

Assumption 3 *The client is configured to use* 20 *resolvers as suggested in [13], i.e we set* $t = 20$*. Our attacks can still be successful for other values of* t*.*

The above general assumptions apply across all our attacks. However, some of the attacks we conduct might require meeting extra conditions. We will clearly highlight any additional assumptions we make.

3.2 DNS Cache Poisoning Attack

The fundamental security objective of DepenDNS is to protect clients from bogus IP addresses received from DNS resolvers. Detecting these bogus IP addresses is based on the calculations performed by algorithm π and the rejection is based on comparing the grade of each IP address to 60. IP addresses having grades, G_i with $G_i \geq 60$ are accepted and are added to A and H. In this attack we attempt to circumvent the scheme by trying to achieve a grade of 60 or higher and eventually inject bogus IP addresses for a host name into the history data of DepenDNS.

Assumption 4 *The history table of DepenDNS contains IP addresses for the host name being requested.*

The attacker's goal is to bypass the security controls implemented by DepenDNS and have algorithm π accept false information in the form of bogus IP addresses. Assumption 4 implies that the value of β is 0 for every bogus IP address, IP_{bogus}, and α_{bogus} is likely to be 0. The reason for this is that n^{bogus} is 1 since the attacker would target a single resolver as per Assumption 2, making it difficult for n^{bogus} to be above the threshold of n^{max}. This leaves the attacker with one variable, γ_{bogus}, to focus on. The attacker needs to make sure that γ_{bogus} for IP_{bogus} is 1. To achieve this, the following conditions must be met:

- The leftmost 16 bit of the bogus IP address is the same as the legitimate IP addresses for the host name, i.e. IP_{bogus} belongs to a valid k^{th} class IP address for the host name being requested.
- $c^k_{current} - c^k_{history}$ is within the pre-defined threshold, i.e. $-0.1 \leq c^k_{current} - c^k_{history} \leq 0.1$

Since the values α_{bogus} and β_{bogus} are 0, then the grade for IP_{bogus} can be calculated as

$$G_{bogus} = \frac{1}{2}(G_{\beta\gamma} + 10 \cdot (N - 1)),$$

assuming γ_{bogus} is 1. For IP_{bogus} to be accepted, the value of G_{bogus} must be 60 or higher, i.e. the following condition must be met:

$$\frac{1}{2}(G_{\beta\gamma} + 10 \cdot (N - 1)) \geq 60.$$

Since $G_{\beta\gamma}$ is 40, then the condition that $N \geq 9$ will guarantee that IP_{bogus} achieves the passing grade.

Our experiments have shown that the value of N is 1 or less for most host names. However, this is not the case when the host name is served by a CDN. We have noticed that the value of N is within a range that would allow attackers to inject bogus IP addresses using the technique we have explained in this section. For example, Figure 1 shows that the average value of N for "www.live.com" is 13.5. We have found similar results for other host names such as "maps.live.com", "www.youtube.com" and "www.vmware.com". The results are shown in Appendix B.

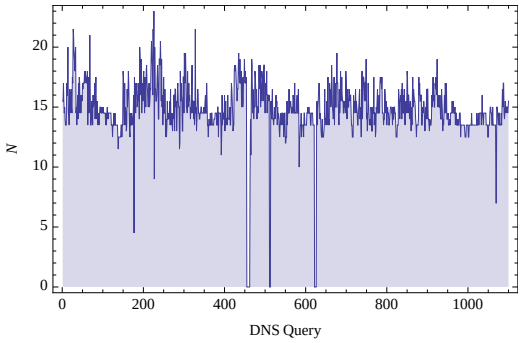


Fig. 1: Value of N over time for www.live.com

Section 4 shows the results of our experiments on DepenDNS. Table 1 provides the percentage of runs when $N \geq 9$. A run is defined as the execution of algorithm π against R and H when a host name is being requested. The table shows that a high percentage of runs had $N \geq 9$ for the host names listed earlier. This gives the attacker an opportunity to launch her attack during the majority of runs. Please note that this is based on real data collected over time and hence includes situations when there are no reply messages due to network connectivity issues causing the value of N to be 0.

Table 1: Percentage of runs with $N \geq 9$

Host name	Number of runs	% of runs with $N \geq 9$
www.live.com	1100	98.3
maps.live.com	1100	98.2
www.youtube.com	1100	98.5
www.vmware.com	1100	80.3
www.hsbc.com	1100	0

We have simulated the attack by injecting a bogus IP address, 96.17.222.222, into the cache of one of the resolvers for the host name "www.live.com". Our attack was successful and the bogus IP address has been accepted by algorithm π and has been added to the history data.

```
...
PASS 0 0 1 100 96.17.222.222 Adding 96.17.222.222 to history
...
```

Impact: *An attacker can inject a bogus IP address that points to a malicious website or inject IP addresses that can make the host being requested unreachable. This is applicable when the host name is hosted by a CDN and the client is running DepenDNS.*

3.3 Denial of Service Attack

Unlike a network based Denial of Service (DoS) attack, our work targets the layer where DepenDNS would operate and where the decision of accepting or rejecting an IP address takes place. In our attack we try to force algorithm π into

rejecting all IP addresses in R for a host name, hence making the host unreachable by clients running DepenDNS. The same spoofing attack scenarios listed in Section 1 can be used by the attacker with the same success probabilities of p_1, p_2 and 1 respectively.

Assumption 5 *The history data of DepenDNS does not contain information about the host name being requested.*

Consider a run of the algorithm π on a set of sets of returned IP addresses R_j, $1 \leq j \leq t$. The above assumption implies that the value of both β_i and γ_i is 0 for each IP_i in R. As a result, $R_\beta = \emptyset$, $R_\beta = \emptyset$ and $G_i = \alpha_i \cdot (G_\alpha - 10 \cdot (N-1))$. For our attack to succeed, all IP_i in R should have a grade value, G_i , of less than 60. Therefore, the following condition must be met to fail each IP_i:

$$\alpha_i \cdot (G_\alpha - 10 \cdot (N-1)) < 60.$$

Since G_α is known to be 60, then N must be higher than 1 for all IP_i to be rejected. N can be calculated as:

$$N = \frac{|R_\alpha|}{\text{Mode}(|R_1|, |R_2|, ..., |R_t|)},$$

since $R_\beta = \emptyset$ and $R_\beta = \emptyset$.

To increase the value of N, an attacker would need to focus on increasing the size of R_α or decrease the modal value of $|R_j|$. Decreasing the modal value proved to be very difficult since we assumed that the attacker would target one resolver only as per Assumption 2.

Our experimental results presented in Section 4.1 shows the value of N for a number of host names queried over a period of time. For example, the average value of N for "www.live.com" is 3.5 causing a self denial of service. Figure 2 shows that no IP addresses have been accepted during the vast majority of runs.

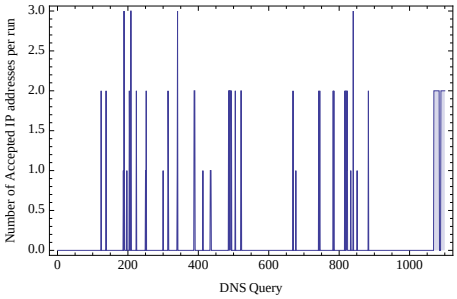


Fig. 2: Number of IP addresses accepted over time for www.live.com

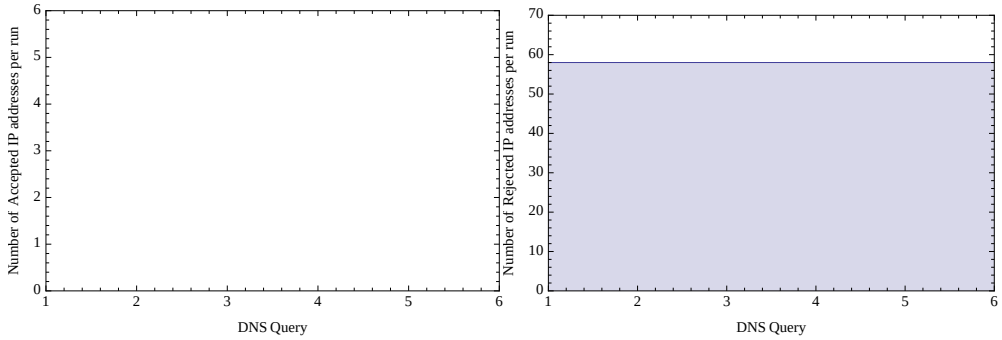
On the other hand, we have noticed that the value of N can be easily influenced in the case when the host name being requested is hosted by a CDN. Therefore, rather than injecting bogus IP addresses in the DNS cache of a resolver, an attacker would include a good number of correct IP addresses that correspond to the host name. The goal is to maximise the number of IP addresses that pass the α test and hence increase the value of $|R_\alpha|$.

We have simulated the attack using real data collected from querying the 20 DNS resolvers for "www.youtube.com" and we have been able to force algorithm π into rejecting all IP addresses received from all the 20 resolvers. We have tested this for six consecutive runs and the attack was successful during each run. Before the attack, a total of six IP addresses would have been accepted as shown in Appendix A. After injecting a number of valid IP addresses as per the technique described in this section we have found that algorithm π starts rejecting all the IP addresses received from the 20 resolvers. Figures 3a and 3b show the number of accepted and rejected IP addresses during the six runs.

Impact: *An attacker can perform a DoS attack against a specific host name when it is hosted by a CDN and when the client is running DepenDNS.*

3.4 Amplification Attack

In this attack we try to exploit the fact that DepenDNS employs a number of resolvers, t. The success of an amplification attack relies on the ability of the attacker to trigger the generation of large volume of traffic by sending requests of negligible size. The higher the amplification factor, the more severe the attack is. Such attacks are not new to DNS. In fact, DNS has been the target of various DNS amplification attacks [6], which rely on the fact that DNS response messages are significantly larger than reply messages causing the consumption of network bandwidth. In practice, an attacker will employ a set of machines under her control, like a botnet, to perform such attacks [6].



(a) Number of IP addresses accepted during the six runs

(b) Number of IP addresses rejected during the six runs

Fig. 3: Results of the attack against "www.youtube.com"

Assumption 6 *In our attack, we take the average size of a DNS request message to be 60 bytes and of a DNS reply message to be 124 bytes. These numbers are based on the data collected during our experiments. This takes into account the TCP/IP headers.*

Our attack uses clients running DepenDNS and does not require the use of a botnet. We show how an implementation of DepenDNS can cause such attacks with a high amplification factor. We also show a sample code for performing the amplification attack.

Our attack is contained in web page's Hyper Text Markup Language (HTML) code. The sample code that we show in this section does not install any software on the client's machine and can be automatically executed by any application which can interpret HTML or JavaScript. The HTML code can be delivered to clients by email or can be published to a website which the client visits. To ensure a large scale effect, the attacker would publish this code on a popular website with thousands of concurrent visitors. Uploading the code onto social networking websites would be an attractive choice to the attacker. The following is an example of an HTML code that employs JavaScript:

```
<html>
<head>
<meta http-equiv="refresh" content="5">
</head>
<body>
<script>
for (i=1;i<=10;i++)
 {
 s1= String.fromCharCode(97+Math.round(Math.random()*25));
 s2= String.fromCharCode(97+Math.round(Math.random()*25));
 document.write('<img src="ftp://'+s1+'.'+s2+'/f">');
 }
 </script>
 </body>
 </html>
```

In the above example, we use the image object, "img", to force the web browser to perform DNS look-up. The size of the above code is 306 bytes. The code generates two random strings, s1 and s2. These strings are then concatenated to build the host name in the image, "img", HTML tag. The attacker can change the number of host names being requested by increasing the number of loops. In the above example, the variable "i" is incremented by one in every loop until reaching 10. Although changing the number of loops in the JavaScript to a higher value has a negligible effect on the size of the code, it has a significant impact on the amplification factor. For example, changing the number of loops to 100 will increase the code size by only 1 byte, but will cause the generation of at least 736 kbytes of DNS request and reply messages taking into consideration Assumption 6. This number will be multiplied by the number of search domains the client is configured for. For example, the expected traffic will be at least 1.58 Mbytes if the client is configured for one search domain such as "example.com".

Impact: *Although, this attack applies to the standard DNS implementation, DepenDNS amplifies it by a factor of*

20 which makes it more attractive to attackers. Hence, an attacker can turn clients running DepenDNS into a source of a serious DoS attack. For example, an attacker can post this code to a popular website causing a storm of DNS traffic on the Internet.

4 Experimental Results

In this section we evaluate the operation of DepenDNS under a number of scenarios using real life data collected over a period of time. We queried 20 resolvers, all located in the US, for the following host names every five minutes, with a total of 1100 queries for each host name:

- "www.live.com". This host name has a CNAME of "a134.g.akamai.net" and is served by a CDN.
- "maps.live.com". This host name has a CNAME of "a1234.g.akamai.net" and is served by a CDN.
- "www.youtube.com". This host name has a CNAME of "youtube-ui.l.google.com" and is served by a CDN.
- "www.vmware.com". This host name has a CNAME of 'e508.g.akamaiedge.net'" and is served by a CDN.
- "www.hsbc.com".

We have developed a perl script that implements DepenDNS. The script takes DNS response messages from the 20 resolvers and runs them through algorithm π. The script also maintains a history table as described in Section 2.1.

We have collected the following set of information for each host name listed above:

- The value of N for each run.
- The number of accepted and rejected unique IP addresses for each run.

A run is defined as the execution of algorithm π against R and H when a host name is being requested.

The results shown in this section validate the findings presented earlier in the paper. We divide our experiments into two categories based on the availability of history information about the host name being requested.

4.1 Experimenting with no History Information

In this section we present the results of running DepenDNS without existing history information about the host name being requested. The results of all the runs show the following trends:

- A large percentage of valid replies are rejected by DepenDNS when the host name being requested is hosted by a CDN. For example, Table 2 shows that 98.6% of the unique IP addresses for "www.live.com" have been rejected after 1100 runs.
- A large number of runs had no accepted IP addresses when the host name being requested is hosted by a CDN. During these runs, the host name being requested is considered unreachable by the client.
- IP addresses for host names that are not hosted by CDNs were accepted in all of the runs.
- The value of N varies depending on the host name being requested.

Table 2 shows the results of running DepenDNS against the five host names. More detailed information about the results can be found in Appendix A. Appendix A presents the value of N over time along with the number of accepted and rejected IP addresses in each run for the host names that we have queried.

Table 2: Summary results for all host names

Host name	After run	Number of distinct accepted IP addresses	%	Number of distinct rejected IP addresses	%
www.live.com	1100	8	1.4	567	98.6
maps.live.com	1100	7	2.7	251	97.3
www.youtube.com	1100	6	5.2	110	94.8
www.vmware.com	1100	16	19.5	66	80.5
www.hsbc.com	1100	1	100	0	0

4.2 Experimenting with Existing History Information

In this section we evaluate DepenDNS when history information exists for the host name being requested. The data used to initialise the history of DepenDNS has been collected at different points of time. We evaluate DepenDNS using history data collected in the following different ways:

- The first set of replies received from the t resolvers.
- The collection of replies received from the t resolvers after one hour.
- The collection of replies received from the t resolvers after twelve hours.
- The collection replies received from the t resolvers after twenty four hours.

The results of all the runs show the following trends:

- A good percentage of valid replies are rejected by DepenDNS. The percentages are listed in Table 3 for the five host names we have queried.
- The value of N is high for host names hosted by CDNs. For example the value of N is around 13.5 for "www.live.com" and 10 for "www.vmware.com".

Table 3: Summary results for all host names

Host name	Age of history	After run	Number of accepted IP address	%	Number of rejected IP address	%
www.live.com	1st run	1100	209	36.3	366	63.7
www.live.com	1 hour	1100	285	49.6	290	50.4
www.live.com	12 hours	1100	342	59.5	233	40.5
www.live.com	24 hours	1100	393	68.3	182	31.7
maps.live.com	1st run	1100	72	27.9	186	72.1
maps.live.com	1 hour	1100	105	40.7	153	59.3
maps.live.com	12 hours	1100	127	49.2	131	50.8
maps.live.com	24 hours	1100	156	60.5	102	39.5
www.youtube.com	1st run	1100	105	90.5	11	9.5
www.youtube.com	1 hour	1100	105	90.5	11	9.5
www.youtube.com	12 hours	1100	105	90.5	11	9.5
www.youtube.com	24 hours	1100	108	93.1	8	6.9
www.vmware.com	1st run	1100	47	57.3	35	42.7
www.vmware.com	1 hour	1100	63	76.8	19	23.2
www.vmware.com	12 hours	1100	70	85.4	12	14.6
www.vmware.com	24 hours	1100	74	90.2	8	9.8
www.hsbc.com	1st run	1100	1	100	0	0
www.hsbc.com	1 hour	1100	1	100	0	0
www.hsbc.com	12 hours	1100	1	100	0	0
www.hsbc.com	24 hours	1100	1	100	0	0

Appendix B presents the value of N over time along with the number of accepted and rejected IP addresses in each run for the host names that we have queried. The values presented in Appendix B are the results of running the scheme using as history that data collected from the first set of replies from the 20 resolvers.

4.3 Location of the t Resolvers

We have also conducted experiments that evaluate DepenDNS when the t resolvers are distributed over multiple geographical locations. The objective has been to compare the results of theses experiments to the ones we have presented earlier. The overall results have shown a lower number of accepted IP addresses in each run. This applies to host names that are served by CDNs. Host names that are not served by CDNs such as "www.hsbc.com" exhibited the same behaviour that we have seen when using DNS resolvers located in the same geography.

5 Conclusion

Proposals which attempt to address challenges in critical infrastructures should carefully study the impact of their implementations. Our analysis of DepenDNS has revealed a set of deficiencies in both the security controls and the

operational related aspects of the scheme. Although the protection controls implemented by DepenDNS have shown to work for general web sites, domains that are hosted by CDNs have proven to be more of a challenge. There are various assumptions made by the proposed scheme that have not been justified nor backed up by scientific evidence. On the other hand, we have found conditions under which denial of service and cache poisoning attacks can be launched against DepenDNS. We have also shown that the implementation of DepenDNS can be exploited by an amplification attack which can cause a large scale unsolicited denial of service attack. As a result, we do not recommend adopting DepenDNS with its current proposed design.

References

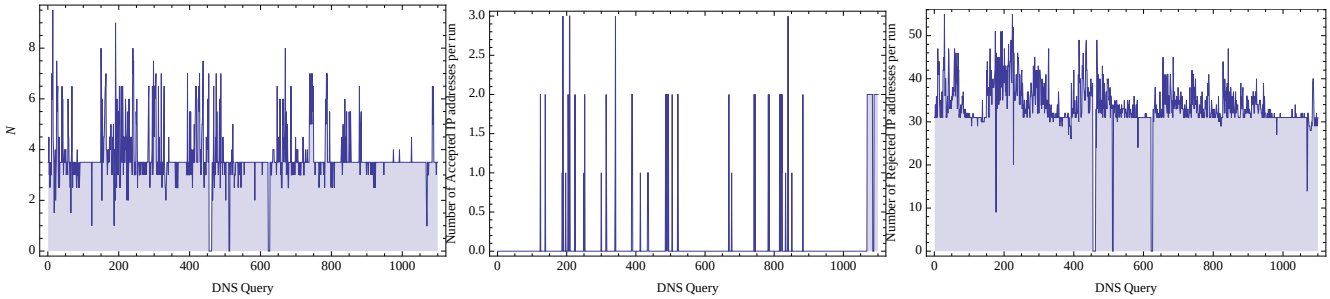
1. D. Eastlake 3rd. Domain Name System Security Extensions. RFC 2535, Internet Engineering Task Force, March 1999.
2. D. Eastlake 3rd and A. Panitz. Reserved Top Level DNS Names. RFC 2606, Internet Engineering Task Force, June 1999.
3. D. Atkins and R. Austein. Threat Analysis of the Domain Name System (DNS). RFC 3833, Internet Engineering Task Force, August 2004.
4. David Dagon, Manos Antonakakis, Paul Vixie, Tatuya Jinmei, and Wenke Lee. Increased DNS forgery resistance through 0x20-bit encoding: security via leet queries. In *ACM Conference on Computer and Communications Security*, pages 211–222, 2008.
5. VeriSign Inc. The domain name industry brief. Technical report, http://www.verisign.com/domain-name-services/domain-information-center/domain-name-resources/domain-name-report-june09.pdf. June 2009.
6. Georgios Kambourakis, Tassos Moschos, Dimitris Geneiatakis, and Stefanos Gritzalis. A fair solution to DNS amplification attacks. In *WDFIA '07: Proceedings of the Second International Workshop on Digital Forensics and Incident Analysis*, pages 38–47, 2007.
7. D. Kaminsky. Its the end of the cache as we know it, 2008.
8. P.V. Mockapetris. Domain names - concepts and facilities. RFC 1034, Internet Engineering Task Force, November 1987.
9. P.V. Mockapetris. Domain names - implementation and specification. RFC 1035, Internet Engineering Task Force, November 1987.
10. Root Servers Technical Operations. http://www.root-servers.org.
11. Lindsey Poole and Vivek S. Pai. ConfiDNS: leveraging scale and history to improve DNS security. In *WORLDS'06: Proceedings of the 3rd conference on USENIX Workshop on Real, Large Distributed Systems*, 2006.
12. K. Stamos, G. Pallis, A. Vakali, and M.D. Dikaiakos. Evaluating the utility of content delivery networks. In *Proceedings of the 4th edition of the UPGRADE-CN workshop on Use of P2P, GRID and agents for the development of content networks*, pages 11–20. ACM New York, NY, USA, 2009.
13. Hung-Min Sun, Wen-Hsuan Chang, Shih-Ying Chang, and Yue-Hsun Lin. DepenDNS: Dependable mechanism against DNS cache poisoning. In *CANS '09: Proceedings of the 8th International Conference on Cryptology and Network Security*, pages 174–188, 2009.
14. Athena Vakali and George Pallis. Content delivery networks: Status and trends. *IEEE Internet Computing*, 7(6):68–74, 2003.
15. Lihua Yuan and Krishna Kant. DoX: A peer-to-peer antidote for DNS cache poisoning attacks. In *IEEE ICC: Proceedings of the International Conference on Communications*, pages 2345–2350, 2006.

A DepenDNS with no History Information

We have collected the following set of information for each host name we evaluated when there is no information available in the history of DepenDNS:

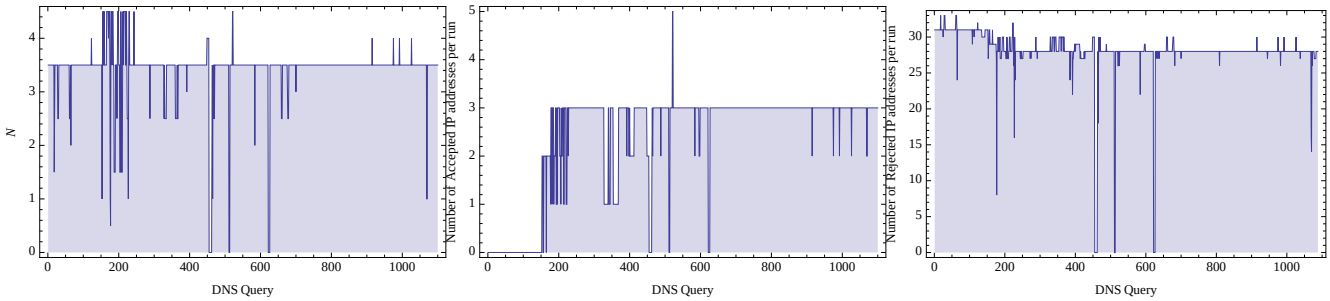
- The value of N for each run.
- The number of accepted and rejected unique IP addresses for each run.

The reader might notice some dips in the graphs shown in the appendix. These are due to loss of network connectivity.



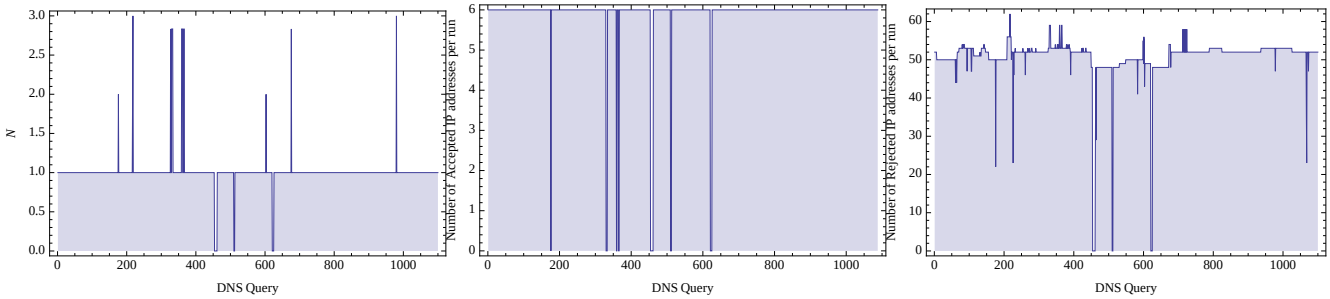
(a) Value of N over time (b) Number of accepted IP addresses (c) Number of rejected IP addresses

Fig. 4: Results for "www.live.com"



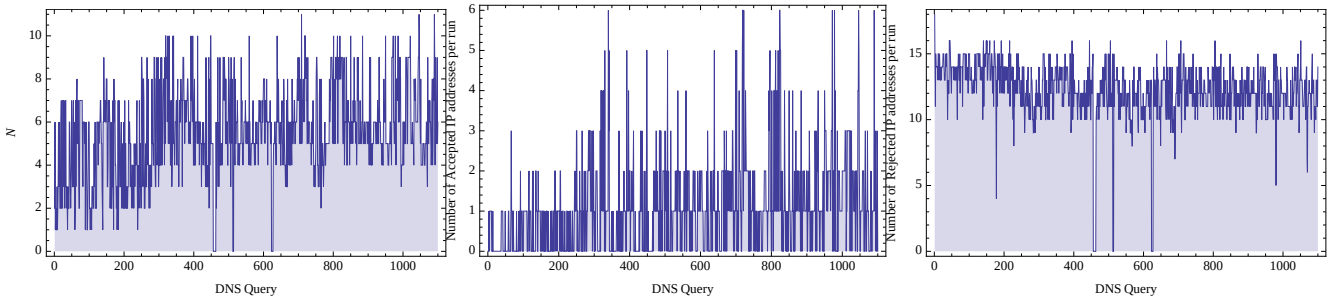
(a) Value of N over time (b) Number of accepted IP addresses (c) Number of rejected IP addresses

Fig. 5: Results for "maps.live.com"



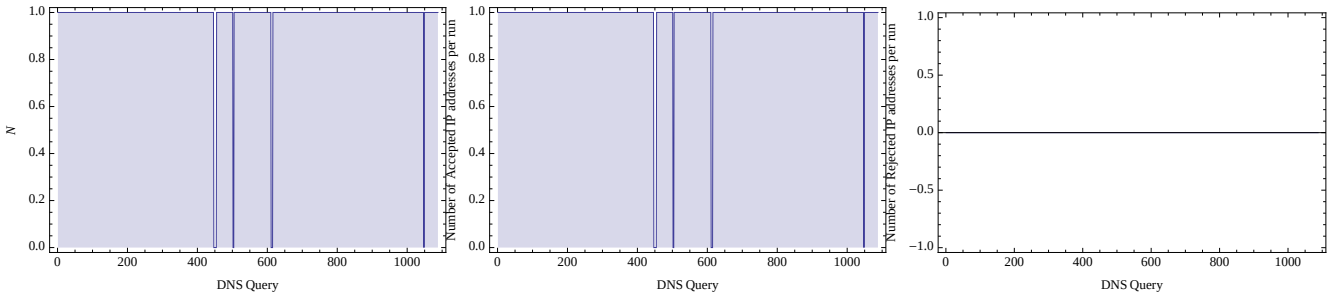
(a) Value of N over time (b) Number of accepted IP addresses (c) Number of rejected IP addresses

Fig. 6: Results for "www.youtube.com"



(a) Value of N over time (b) Number of accepted IP addresses (c) Number of rejected IP addresses

Fig. 7: Results for "www.vmware.com"



(a) Value of N over time (b) Number of accepted IP addresses (c) Number of rejected IP addresses

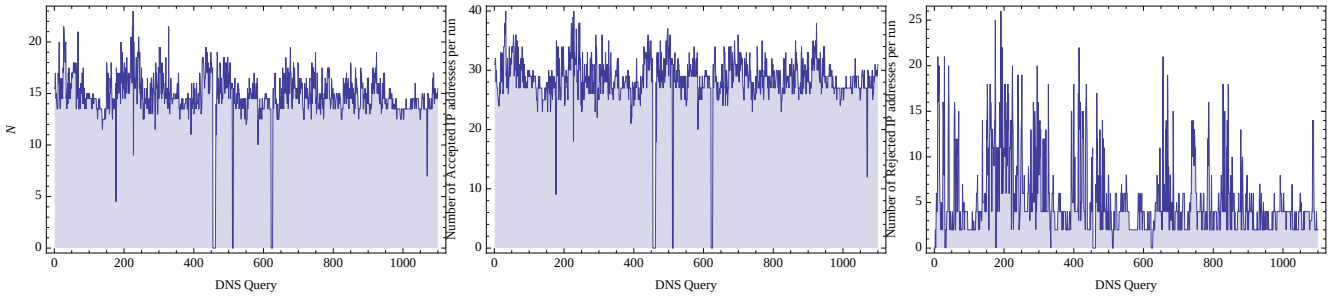
Fig. 8: Results for "www.hsbc.com"

B DepenDNS with with Existing History Information

We have collected the following set of information for each host name we evaluated when the history of DepenDNS is populated from the information received as a result of the first DNS resolution requests sent to the t resolvers.

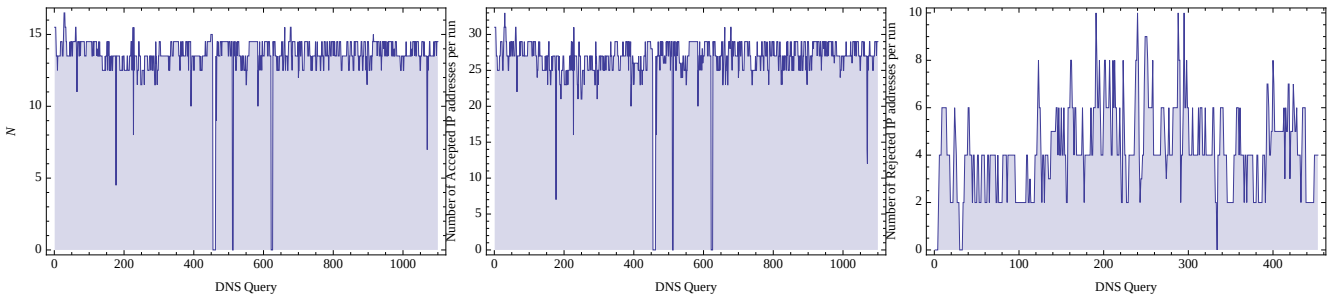
- The value of N for each run.
- The number of accepted and rejected unique IP addresses for each run.

The reader might notice some dips in the graphs shown in the appendix. These are due to loss of network connectivity.



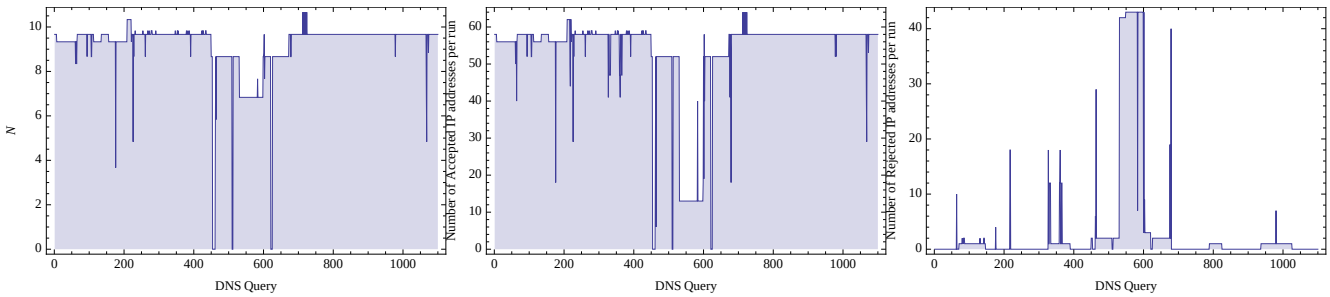
(a) Value of N over time (b) Number of accepted IP addresses (c) Number of rejected IP addresses

Fig. 9: Results for "www.live.com"



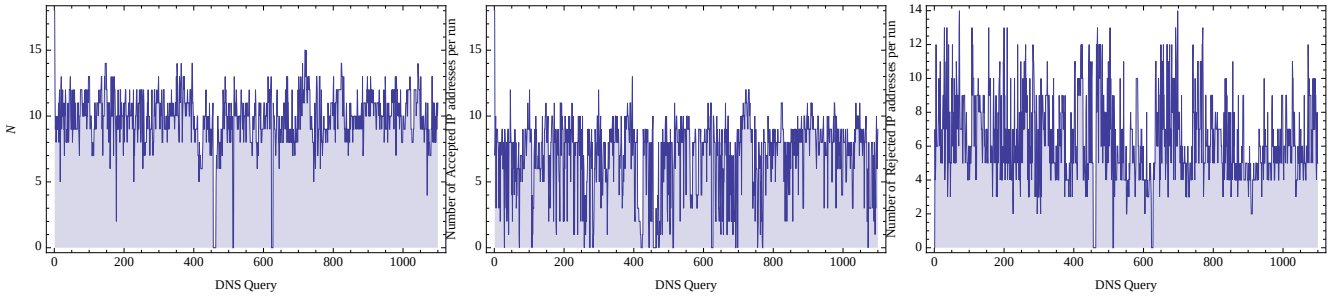
(a) Value of N over time (b) Number of accepted IP addresses (c) Number of rejected IP addresses

Fig. 10: Results for "www.maps.com"



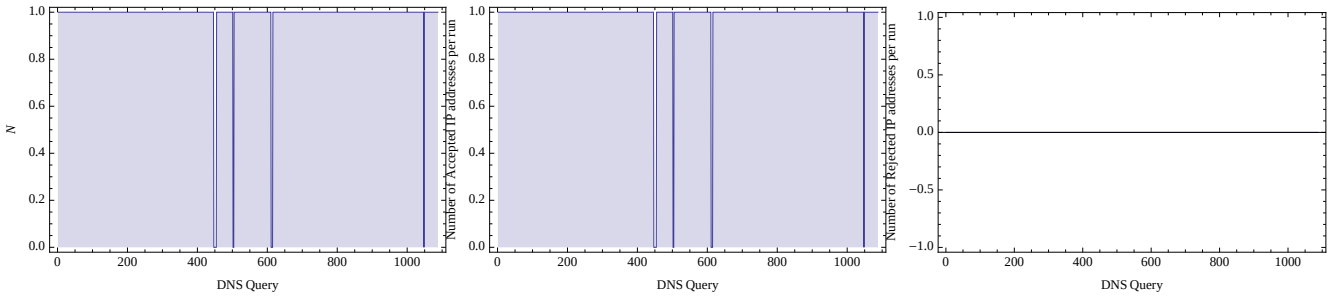
(a) Value of N over time (b) Number of accepted IP addresses (c) Number of rejected IP addresses

Fig. 11: Results for "www.youtube.com"



(a) Value of N over time (b) Number of accepted IP addresses (c) Number of rejected IP addresses

Fig. 12: Results for "www.vmware.com"



(a) Value of N over time (b) Number of accepted IP addresses (c) Number of rejected IP addresses

Fig. 13: Results for "www.hsbc.com"