
Bachelor Thesis

# SoS: Stub (Resolvers) on Steroids

by

Roy de Regt
(rrt520)

*Submitted in partial fulfillment of the requirements*
*for the degree of*
*Bachelor of Science*
*in*
*Computer Science*
*at the*
*Vrije Universiteit Amsterdam*

August 22, 2021

Certified by .....
Balakrishnan Chandrasekaran
Assistant Professor
*First Supervisor*

Certified by .....
Balakrishnan Chandrasekaran
Assistant Professor
*Daily Supervisor*

Certified by .....
Herbert Bos
Full Professor
*Second reader*

# SoS: Stub (Resolvers) on Steroids

Roy de Regt
Vrije Universiteit Amsterdam
Amsterdam, NL
r.b.de.regt@student.vu.nl

## ABSTRACT

The DNS protocol can be easily abused to obtain sensitive details about end-users' Internet activities. Research on these privacy attacks have improved every single component of the DNS with one exception—the stub resolver. In this thesis, we simply ask, "what is the role of the stub resolver in safeguarding end-users' privacy?"

We first address the crucial performance-oriented question that has perhaps stymied prior work from improving the functionality of stub resolvers: To what extent does adding functionality to stub resolvers affect end-user's browsing experiences? To this end, we develop stub resolvers with various small adaptations and observe how they fare relative to a vanilla stub resolver, which relies entirely on a recursive resolver for resolving names. We show that even when stub resolvers perform resolutions on their own, page-load times of web pages can perform similar to the vanilla stub resolver. We then analyze the feasibility of augmenting stub resolvers with additional functionality to improve end-user privacy and discuss two key ideas.

## 1 INTRODUCTION

The Domain Name System (DNS) [26, 27] is a fundamental component of the Internet. Every single request for a resource on the Internet typically begins with a DNS request. Despite the ubiquity of DNS, security was not considered in its design, as was the case with many other Internet protocols. Back when these protocols were defined, the Internet was quite simple—with few routers and hosts—and there was implicit trust among all these entities. Today attacks are rampant and data breaches are commonplace. Given the plethora of online services that we use today to carry out many mundane as well as crucial everyday tasks, we are in dire need of protecting the privacy of end users. More concretely, except for the two parties involved in a web request, viz., the end user and the web site serving the content, no other entity should be able to determine which end user requested what content.

One of the key design issues in DNS that facilitates an attacker to eavesdrop on the conversation (i.e., the DNS resolution process) is that the traffic, per the original design [27] is transmitted as plaintext. Over the years, many improvements to the DNS have been made in order to mitigate vulnerabilities that impact the user's privacy. Recursive resolvers and authoritative name servers have adopted encryption protocols [20, 23, 30] and data verification techniques [1]. Yet, the only component in the DNS ecosystem that has not seen any improvement is the one on the user's machine—the stub resolver.

The stub today simply delegates its work to the local resolver. The local resolver is able to observe and monitor all incoming queries, revealing users' IP addresses and online behaviour [7, 29]. This surveillance—or even the fact that this surveillance is possible—is an attack on the user's privacy [11, 35]. DNS lookups can, however, be performed by the stub resolver itself—without any help from the recursive resolver. Perhaps conventional wisdom dictates that the stub resolver cannot provide security against privacy attacks as augmenting the stub with additional functionality will introduce significant performance overheads.

In this thesis, we explore options to adapt the stub resolver to improve end-user's privacy. To this end, we answer the following questions. What are the performance implications of making a stub resolver perform recursive resolutions on its own—without delegating to a recursive resolver? To what extent do such performance overheads matter? More concretely, do these overheads impair, for instance, end-users' web-browsing experiences? How can we augment the functionality provided by the stub resolver to improve the privacy of end users? More succintly, we look for ways to change the stub resolver to improve the end-user's privacy while preserving performance.

We augment stub resolvers' functionality and refer to these augmented resolvers as stub (resolvers) on steroids, or SoS resolvers in short. The SoS resolvers do not rely on recursive resolvers, removing the recursive resolver's ability to perform pervasive monitoring [11] of end users. We measure the overheads our augmentations introduce when resolving a domain name and their implications for web (browsing) performance. We present feasibility studies of two approaches for improving end-users' privacy. In the first, we adapt a prior work that shields users' queries from untrusted name servers by making the stub resolve a name along with `n - 1` "dummy " domains to hide the actual query. In the second, we suggest an encryption scheme that allows for efficient, connectionless DNS encryption from the stub. By encrypting DNS traffic the end user is protected against on-path observers' eavesdropping.

## 2 THREATS TO DNS PRIVACY

Typically, a successful DNS lookup works as follows: A stub resolver (or, stub, in short) resides in the user's machine. When an application requires a name to be resolved, the stub sends a query to the local resolver. The local resolver does the complete recursive resolution of the domain name and returns a query response to the stub, by retrieving an answer from cache or querying one or more authorative name servers until it has the answer for the query. This local resolver is typically provided by the user's Internet Service Provider (ISP). Originally, stubs were to do the recursive lookup themselves [26], until some large networks started funneling DNS requests to a local resolver, creating a shared cache [12].

On-path observers can eavesdrop users' queries since the queries are in unencrypted form. A local resolver aggregates the resolutions for hundreds of thousands or even millions of users. This aggregation reduces the diversity of path that DNS query packets might otherwise take. Using a stub resolver, hence, improves the path

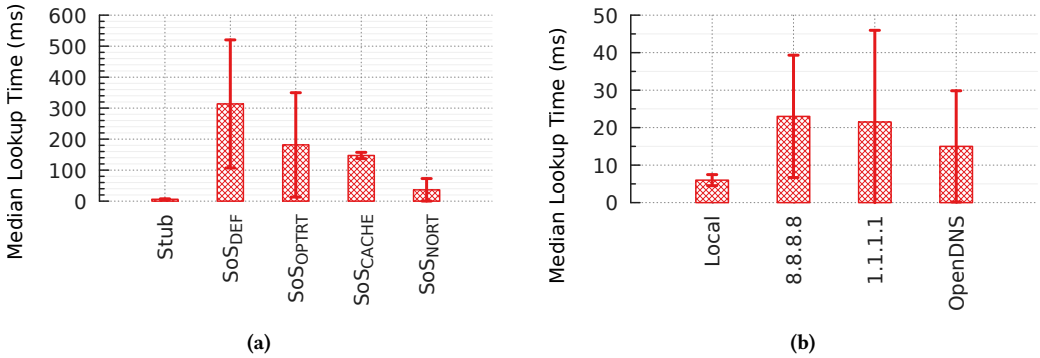



**Figure 1:** ***(a) SoS resolvers have larger lookup times than the stub resolver. The SoS$_{NORT}$ performs best among SoS resolvers. (b) When a stub resolver uses open resolvers rather than the local resolver, we observe longer lookup times.***

diversity (from the end users to various authoritative nameservers) and likely makes it hard for an attacker to eavesdrop. Eavesdropping on end-users DNS queries is still not impossible even when using SoS, since the network paths between the end users and the authoritative nameservers might overlap significantly. We, therefore, present an option to encrypt the DNS queries and responses by leveraging DNS records already deployed by the widely used DNSSEC implementations.

Aside from on-path observers, the recipient of the query can also observe all incoming queries. The local resolver can therefore log and monitor all queries made by their users, revealing privacy-sensitive information about users in the process. An ISP can thus pervasively monitor users' behaviors (inferred from their DNS queries), thereby constituting an attack on privacy [17]. Today, users could turn to Trusted Recursive Resolvers (TRRs). A TRR is a third party other than the ISP recursively resolving queries for their users. TRRs can be hosted by a third party that also hosts Content Distribution Networks (CDNs). If the user requests a domain that is hosted on a CDN owned by the TRR, the TRR is capable of returning a result that is hosted on a CDN close to the user. A TRR can also provide the user with options for encryption of queries and responses. Rather than solving the problem, it merely shifts the trust to another organization. We aim to defend the user against pervasive monitoring by any recursive resolver by recursively resolving domain names from the stub itself.

Recent work provides new standards for encrypted information exchange between stubs and recursive resolvers [20, 23, 30]. The encrypted DNS queries and responses prevent anyone on the path from stub to recursive resolver and vice versa from pervasive monitoring [17] and DNS hijacking [36]. The most widely implemented DNS encryption protocols are DNS over HTTPS (DoH) and DNS over TLS (DoT). Both DoT and DoH rely on a TLS connection. TLS connections have a performance overhead from the initial handshake, that is amortized over multiple queries to the same recursive resolver [22]. Using the same connection (and therefore, the same recursive resolver) for all outbound DNS queries leads to centralization of the DNS [21], introducing new privacy concerns [24].

Domain Name System Security Extensions [1] (DNSSEC) is a set of specifications that extend the DNS protocol. It uses a public key infrastructure (PKI) for the signing of resource records such that integrity is provided. Stubs and recursive resolvers can use the PKI to verify that a received answer has not been tampered with.

## 3 SUPERCHARGING THE STUBS

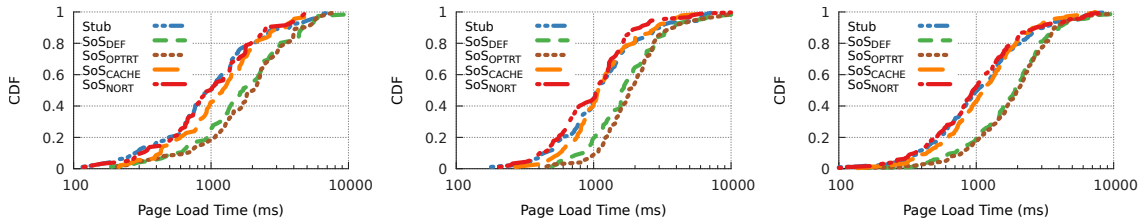
From an end-user's perspective, there's no benefit to relying on a recursive resolver to do the DNS resolution besides a potentially shorter resolution time. This benefit stems from the nature of the recursive resolver: most recursive resolvers serve many users. When multiple different users ask for the same domain name (roughly around the same time), the resolver can do the resolution once, cache the record, and then serve that record from cache. That is, as long as there is more than one request for this domain within the time to live (TTL).

The performance benefits of a shared recursive resolver might be smaller than one would expect. Prior work has showed that in a data set of 200 million DNS queries measured over 14 months, 70% of DNS responses have TTL values of at most 3600 seconds, or one hour [9]. Nevertheless, less than 0.01% of domains are queried more than 10000 times, or more than once an hour on average during this 14-month period. Another analysis of DNS traffic notes that in their dataset, 77% of all observed domains are only ever queried by a single user [33]. These observations suggest that for the majority of domains, no-one benefits from the caching of a resource record at the recursive resolver.

Resolving domain names from the stub resolver is not as bad as it seems, if the benefit of caching is small. Though, one has to be careful of performance overheads. To evaluate the performance impact of augmenting stub resolvers with new features, we systematically add new features to the stubs and measure the performance overheads they introduce. We compare the performance of SoS resolvers to that of the stub that queries a recursive resolver. Specifically, we measure the impact of the new features on individual query-lookup times as well as the overall page-load times.

To prevent a recursive resolver from monitoring users' web behaviour, we create various SoS resolvers that do not rely on recursive resolvers. Each SoS has a unique feature that impacts their name resolution strategy.

- SoS$_{DEF}$: Stub resolver that queries a root name server for the TLDs it requires and follows the DNS hierarchy from there.




**(a)** ***Hispar$_{top\ 100}$*** **(b)** ***Hispar$_{bottom\ 100}$*** **(c)** ***300 random pages, with a 1:5 ratio of landing to internal pages.***

**Figure 2:** ***Measured Page Load Times of stubs and SoS resolvers with different sets of domains. SoS$_{DEF}$ and SoS$_{OPTRT}$ consistently perform worse than the stub and other SoS resolvers. Simple features to extend the SoS resolver, such as caching or eliminating the round trip time to the root name server, can improve performance to a level where it is comparable of that of the stub resolver.***

- SoS$_{CACHE}$: Stub resolver that caches all query responses.
- SoS$_{OPTRT}$: Stub resolver that queries the root name server with the lowest round trip time (RTT).
- SoS$_{NORT}$: Stub resolver equipped with the root zone file.

SoS$_{DEF}$ will follow the entire DNS hierarchy, starting at a random root name server. SoS$_{OPTRT}$ will also follow the entire DNS hierarchy, but will start at the root name server with the lowest RTT. SoS$_{CACHE}$ will follow the entire DNS hierarchy, unless it has required records cached. For example, records of commonly used TLD name servers are likely to be cached after multiple lookups. Therefore, the SoS$_{CACHE}$ reduces the amount of round trips to the root name server. SoS$_{NORT}$ knows the contents of the root zone file, and can therefore eliminate all queries to the root name server.

The remainder of this section will evaluate the features of the SoS resolvers. To observe the performance impact of resolving a name through a SoS resolver, we measure lookup times from 500 random unique domains with the stub and the SoS resolvers. Furthermore, we measure the impact of using SoS resolvers on the user experience by measuring Page Load Time (PLT) and Speed Index (SI) of different sets of domains. The URLs are gathered from Hispar, a recent work on web performance and evaluation, which discussed what web pages to select for evaluating the effect of web performance on end-users' browsing experiences [5]. This dataset is split in multiple sets of URLs: a set with one URL for every domain in Hispar$_{top\ 100}$, a set with one URL for every domain in Hispar$_{bottom\ 100}$, and a set with 300 random URLs with a mixture of landing and internal pages in the ratio of 1 to 5. We choose to seperate Hispar$_{top\ 100}$ and Hispar$_{bottom\ 100}$ as we expect a difference in cache hit rates: pages in the first are more popular than pages in the latter, and are likely queried more often. Therefore, we expect the probability of a cache hit rate to be higher for pages in Hispar$_{top\ 100}$ than for pages in Hispar$_{bottom\ 100}$. The measurements become more realistic through the use of a combination of landing and internal pages [4].

## 3.1 DNS Lookup Times

Initially, we will turn our focus to two questions: (a) *How does the performance of SoS resolvers compare to that of stubs?* and (b) *How can we rank order the features in terms of performance?*

Figure 1a shows that SoS resolvers all have larger median lookup times than a stub. Equipping the SoS with a root hints file with nearby root name servers reduces the median lookup times. This is explained by the reduced round trip time to the root name servers. Using caching in the stub resolver results in a lower median lookup time than the stub$_{local\ root}$. Eliminating the root name server results in the shortest median lookup times for an SoS. This can be explained by the reduced number of round trips as there is no query to the root name server.

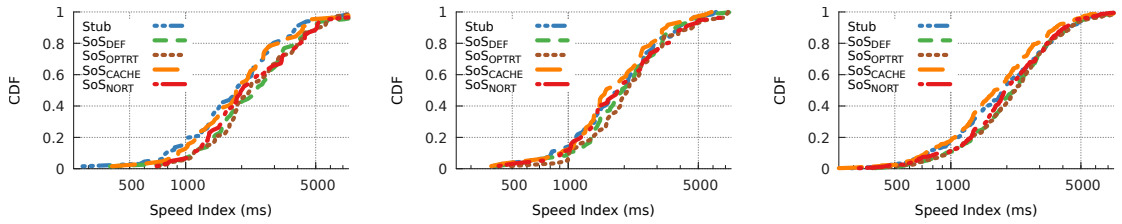
Figure 1b shows that the local resolver's lookups are faster in the median than public resolvers, even though their function is equal: a local resolver serves the same task as a public resolver. The local resolver's lookup times also have a lower median absolute deviation than the public resolvers'. Both the greater median and greater variance are likely explained by larger round trip times to the public resolvers. Public resolvers are typically not in the client's network and the local resolver typically is, since the local resolver is ran by the ISP and the public resolver is not.

## 3.2 Page Load Time and Speed Index

SoS resolvers have larger lookup times than stub resolvers. However, the average web page load consists of much more than a DNS lookup. We now shift our attention to the question: *How does using a SoS resolver, rather than the stub resolver, impact the user experience?*

We define the PLT as the time between navigating to a page and the rendering of the first pixel of the web page. We use the `navigationStart` and `firstPaint` events to determine the PLT values. We use an existing framework to measure PLTs [16]. The figures in 2 show that the augmentations of the SoS with two features, caching and root elimination, makes the performance comparable to that of the stub resolver. We also observe that using SoS$_{DEF}$ and SoS$_{OPTRT}$ will result in a a larger performance loss than the other SoS resolvers. SoS-resolvers require an optimization feature to lower or negate the performance impact, compared to the stub. In 2a, the distribution of PLTs when using SoS$_{NORT}$ differs little from that of the stub resolver. In 2b, we observe that PLTs when using SoS$_{NORT}$ are, for the majority of the distribution, lower than that of the stub resolver. Since the domains in Hispar$_{bottom\ 100}$ are less popular than those in Hispar$_{top\ 100}$, the probability of a cache hit at the local resolver is lower in 2b than in 2a.

PLT is not always a representative metric for the rendering time of a web page [16]. Hence, we also measured the SI scores of the same set of web pages, using Google Lighthouse [18]. The SI




(a) *Hispar$_{top\ 100}$*

(b) *Hispar$_{bottom\ 100}$*

(c) *300 random pages, with a 1:5 ratio of landing to internal pages.*

**Figure 3:** ***Measured Speed Index of stubs and SoS resolvers with different sets of domains. SoS$_{CACHE}$ has a lower median Speed Index than other SoS resolvers in (a), (b), and (c), and a lower median Speed Index than the stub in (b) and (c). Interestingly, the general trend of most SoS resolvers follow that of the stub resolver.***

depends on how quickly the contents of a page are visibly populated [19]. The lower the SI score, the faster the page renders. The figures in 3 show that small optimizations increase performance in web browsing. Similar to 2, 3 shows that SoS$_{DEF}$ and SoS$_{OPTRT}$ perform worse than the other SoS-resolvers, emphasizing the gain by use of the features in SoS$_{NORT}$ and SoS$_{CACHE}$. In 2a we observe the stub resolver slightly outperforming the SoS resolvers. The resolver that is being used has, however, only minimal impact on the speed index. The trend of all SoS resolvers show much similarity to that of the stub resolver. When we either add a cache or remove the round trip to the root name server, there is little difference in web browsing performance between an SoS resolver and a stub resolver.

### 3.3 Impact of caching in stub resolver

Like recursive resolvers, stubs can maintain a cache of answers. This is beneficial for SoS resolvers for two reasons. First, while a user navigates a website, an answer may be needed multiple times in quick succession. Having these answers in cache means there is no need to look them up. Second, when multiple SLDs from the same TLD are queried, the TLD's resource record gets cached. The following queries to that TLD will not require a query to the root name server for this TLD, reducing the number of round trips.

To observe the impact of caching answers with repeated queries, we query 500 domains twice in a short time span with the stub resolver. We observe both lookup times for every domain queried. The first query is with a "cold" cache, the domain is not in the local cache. Hence, the stub should follow the DNS hierarchy to obtain the resource record with the requested domain. The stub resolver will cache this resource record and the response to following queries will come from this cache. By measuring query latencies of both queries to the same domain, we can see the impact of the caching by the stub resolver.

To analyze the impact of having TLD's resource records in cache, we perform DNS lookups of 500 domains with both SoS$_{DEF}$ resolver and a SoS$_{CACHE}$. SoS$_{DEF}$ uses the root name server to obtain the TLD's resource records for every domain that is resolved. SoS$_{CACHE}$ keeps the TLD resource records in cache until the TTL expires. We observe the query latencies from both stub resolvers when querying the same 500 domains.

Figure 4a shows that repeatedly querying the same domain takes less than one millisecond in the second lookup. This is because this record can be obtained from the local cache. In figure 4b we observe that caching results is even beneficial when no domain is queried more than once. This is likely the result of the caching of commonly required answers such as TLD name server records.

## 4 STUBS ON STEROIDS

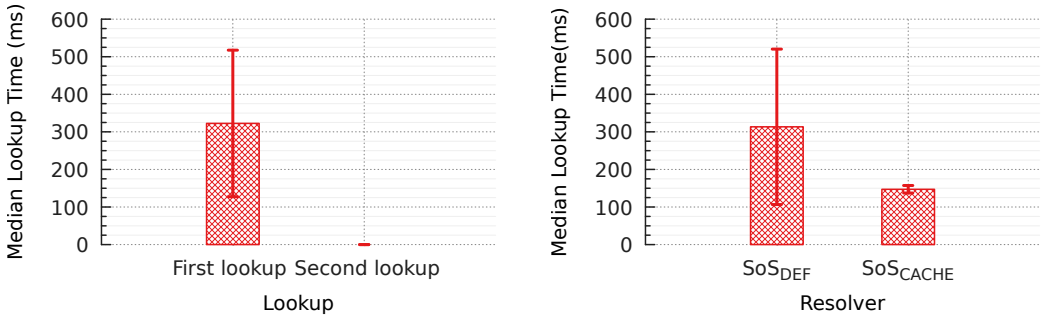
In the previous section, we find that SoS resolvers can be used without much performance loss for the end-user. In this section, we explore the feasibility of two privacy features that can be used in the SoS resolvers.

### 4.1 Masking the Query

In a paged DNS [6], a requester does not query a single domain, but rather queries a page. A page consists of a large number of records—including the one the requester needs. Such pages are created and updated by the TLD name servers. The "dummy" records camouflage the one true record that is needed, protecting against pervasive monitoring by the receiver and any on-path observers of the DNS transaction. Asoni et al. propose a paged DNS between the local resolver and the TLD name servers, preventing the TLD name servers from observing which domains are looked up and whether there is a certain interest for a certain domain. The local resolver computes a hash from the queried domain. That hash is used as a page identifier. The TLD name servers maintain a set of pages, such that all TLD name servers will return the exact same page when queried a certain page identifier. Maintaining pages requires changes to the DNS hierarchy. The proposal on a paged DNS highlights exchange of pages between local resolvers and TLD name servers. Given the results in §3, we examine the exchange of pages between the stub and some resolver.

We evaluate the feasibility of a paged DNS in a SoS resolver by observing the performance overhead of querying pages from the stub. We assume that there is some resolver that can provide the stub with the required page, upon querying a certain page identifier.

We argue that this system is impossible to realize without sacrificing performance with current computing power and network bandwidths. In an average scenario, the user will take a large performance hit from downloading complete DNS pages. A page with 10000 records, is 1MB in the mean [6]. On mobile phones for instance, using a 4G connection with an average 4G downloading bandwidth of 20 Mbps [2] will result in a DNS page downloading




**(a)** ***Lookup times of querying 500 unique domains twice*** **(b)** ***Lookup times of querying 500 unique domains***

**Figure 4:** ***(a) Repeated look ups have a lookup time of 0 ms. (b) The SoS$_{CACHE}$ has a lower median lookup time than the SoS$_{DEF}$.***

time of nearly half a second. For a web browsing session, this will lead to larger PLTs and less user satisfaction [15]. Frequently, web pages require multiple DNS lookups as not all of a web page's content is stored at a single web server. A stub resolver using a paged DNS will therefore be likely to query multiple pages for a single web page visit. Querying a certain set of pages may reveal information about the user's browsing behaviour, requiring the stub to send out queries for 'dummy' pages as well. Querying multiple pages will lead to large PLTs. Another issue is that of memory requirements: pages should be cached at the stub to improve performance and to ensure no patterns in repeated page requests occur. Storing a large number of 1 MB pages will result in the use of a relatively large amount of memory.

## 4.2 Encrypting the Query

Encryption of DNS traffic is the most commonly used DNS privacy defense. The most commonly used encryption protocols, DoH and DoT, rely on a TLS connection for secure transmission of DNS payload. TLS connections have an overhead from the initial handshake, introducing a relatively large lookup time for the first query [22]. An SoS specifically does not forward all traffic to either one or few recursive resolvers. Therefore, the handshake overhead from TLS-based protocols will result in slow lookup times when used with an SoS. To encrypt DNS traffic from an SoS efficiently, we require a connectionless encryption scheme.

DNSCurve [14] uses connectionless encryption of DNS packets with elliptic curve cryptography. We suggest an adaptation to this scheme, where each stub and each root zone have a private/public key pair. Stubs obtain the root zone's public key from the parent zone. This key, combined with the stub's private key, can be used to determine a shared secret that a TLD name server can also determine with the zone's private key and the stub's public key.

A problem in this scheme is that a single DNS server can be authoritative for multiple zones. Using a single key pair for one authoratitive DNS server forms a greater security risk than using key pairs for all zones, as a breach of one zone would lead to a breach of all zones under this authoratitive name server. To tackle this issue, the encrypted query is appended with the zone name, before this package is encrypted with an Authorative Server Key (ASK) that is used by this server for any incoming query, regardless of the zone. The server can decrypt the package with their ASK, observe the zone name, and use the corresponding ZSK to decrypt the query. The stub appends their public key to the encrypted query. The DNS server can now use their private key and the stub's public key to obtain the shared secret needed to decrypt the query. The DNS server uses the same shared secret to encrypt the response, which can later be decrypted by the stub. This scheme is depicted in figure 5.

To distribute name server's public keys to stubs, we can use the existing DNSSEC record types. This allows for easy adoption: authoritative name servers can simply ask the child for their ASK and ZSK, and store them as DS or DNSKEY records. It also allows for backward compatibility with both stubs and servers that do not support this scheme: stubs can still send unencrypted queries to the name servers that may or may not support the encryption scheme.

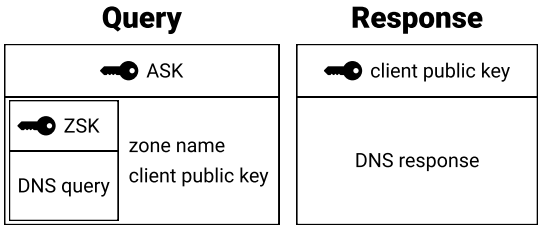



**Figure 5:** ***DNS encryption from the stub. The stub encrypts the query twice, with symmetrical keys obtained from their own private key and the ZSK and ASK public keys. Name servers decrypt using the ASK private key and the stub's public key, read the zone name, and use the corresponding ZSK with the stub's public key to decrypt the query itself. The name server encrypts its responses using the ASK and the stub's public key. The stub knows which zone the response is coming from, and therefore knows which symmetrical key to use for decryption of the response.***

# 5 DISCUSSION

Using an SoS may not completely protect against pervasive monitoring, it does however protect against the pervasive monitoring

by the recursive resolver. When combined with the encryption scheme in §4.2, users are also protected against on-path observers' eavesdropping. Name servers can observe queries for their database entries, rather than a third party observing queries for database entries in a name server elsewhere. A second benefit here is that we re-decentralize the DNS. Queries are distributed over the name servers, rather than to a single recursive resolver, re-decentralizing the DNS ecosystem. A re-decentralized DNS leads to a more robust system, since a single point of failure (the recursiver resolver) will not kill access to the entire ecosystem for all users of that recursive resolver.

The encryption scheme in §4.2 introduces extra database entries for public key distribution, leading to larger storage requirements at the authoratitive name servers. A resource record containing a 256-bit X25519 public key is, depending on the QNAME length, roughly 100 bytes. We argue that the overhead of one RR per zone and one RR per authoritative name server is acceptable since the scheme obviates the need for RRsig records in DNSSEC. RRsig records are obsolete in our encryption scheme, as they are used for integrity. Integrity is guaranteed through the encryption with the shared secret. Both stub and server will see a computational cost for cryptographic operations too. For end-user's devices, the largest potential problem is that it might impact battery life. Earlier work shows that the impact of cryptographic operations in HTTPS have no significant impact on the battery life of a mobile phone [28]. Since most name servers support DoT, they are prepared for an overhead of cryptographic operations. The impact of the cryptographic operations on lookup times is negligible: in our simulation we measure a median overhead of 355 ns. Since there is no longer a need for RRsig records with this scheme, we eliminate the computational overheads from signing records, sending signatures and storing RRsig records. Assuming that a user would use DoT if they were not using this encryption scheme, there is also no longer a need to keep connections alive, reducing memory and network impact.

Moving the load from the shared resolver to the stub will lead to more traffic at the name servers. Especially TLDs will see more DNS traffic, as they tend to be authoratitive for a large number of SLDs. Previous work has shown that indeed the load at the name servers will increase, but the associated computing and performance costs are modest [32].

Hiding what is being queried rather than hiding the sender of the query can also provide privacy. In the current state of technology the two most promising solutions of hiding a query, paged DNS and PIR, are simply too performance-costly.

One can argue that to hide the user's IP address, the user can simply hide behind a proxy. However, this requires trust in the proxy, leading to a similar situation as with the use of a TRR.

# 6 RELATED WORK

In this section, we discuss other available approaches of privacy improvements of the DNS. We group the related work in two types of protocols: protocols that hide what is being queried, and protocols that hide the sender of a certain query.

## 6.1 Hiding the Query

QNAME Minimization [8] focuses on minimizing the amount of data a resolver sends to the authoritative name server. Rather than querying the root name server for a complete domain (e.g. "www.example.com"), a resolver using QNAME Minimization will query ".com". This provides the user some privacy: in the given example the root name server will not be able to see the full QNAME that is queried. QNAME Minimization has the potential to reduce the pervasive monitoring of queries, but does not fully mitigate it. Besides, it can have a significant impact on DNS performance [13].

A paged DNS is suggested to prevent large-scale information gathering at the DNS root and the Top Level Domain (TLD) registries [6]. In a paged DNS, TLDs store pages of 10000 resource records (RRs). Rather than querying a single RR, the local resolver queries one page that contains the required RR to obfiscate the exact RR that the local resolver wants to resolve. When a proper paging system is used, identifying the required RR from a given query is nearly impossible [6]. A Paged DNS does not provide any protection against pervasive monitoring by the local resolver. Hence, we made a case study of querying pages from the stub. We concluded that this is too performance-costly.

An alternative approach to hiding queries is a Private Information Retrieval (PIR) protocol [10]. PIR schemes allow a user to retrieve an item from a server's database, without revealing to the server which item is retrieved. In the DNS, this would protect the user from pervasive monitoring by resolvers they are querying as the server would not be able to know what is being queried.

## 6.2 Hiding the Querier

Oblivious DNS (ODNS) [31] introduces end-to-end encryption of queries and responses between stub and ODNS resolver, with a recursive resolver in between to hide the stub's IP address. A query is forwarded to the ODNS resolver, which decrypts the query and recursively resolves the queried domain name. The ODNS resolver encrypts the response and returns it to the recursive resolver. Finally, the stub decrypts the response. ODNS fails to completely protect the user's privacy: a collusion between recursive resolvers and ODNS servers or authorative name servers will break confidentiality. This is not an unrealistic scenario since a recursive resolver can decide which ODNS server to query. This might be their own ODNS server.

Another adaptation of the current DNS is distributed DNS (D-DNS) [21], in which an adapted stub distributes queries over multiple recursive resolvers rather than sending them to one specific recursive resolver. D-DNS can reduce the amount of information gathered about the end user by a single recursive resolver. Instead, fractions of all queries are spread over a multitude of resolvers.

Elimination of the root name servers is suggested [3] to simplify the DNS ecosystem. Keeping a local copy of the root zone file at the recursive resolver removes a layer from the DNS, resulting in simpler and less resource-intensive DNS infrastructure [3]. While this development is aimed towards the use in recursive resolvers, an SoS benefits from root elimination as well, as it reduces the number of round trips for a name resolution.

Oblivious DNS over HTTPS (ODoH) [25, 34] uses a double proxy rather than a recursive resolver to hide user's IP addresses from

ODNS servers. A stub can choose which ODNS server they query, reducing the power of the recursive resolver to determine the ODNS server used. This does not mitigate association attacks where the proxy and the ODNS server collude.

### 6.3 Future Work

Future work can be aimed at the development of efficient PIR algorithms, as they can be used to provide the user privacy guarantees in the DNS. Alternatively, research can be put towards hiding the user's identity by obfiscating the IP address, without need for trust in some third party to do this for the user. Lastly, work can be put into discovering more exact requirements for a paged DNS system used from the stub resolver. Perhaps smaller pages can provide sufficient privacy, or users' bandwidth will keep increasing to the point where the user will not see any significant impact from querying pages from the stub.

## 7 CONCLUSION

The possibility of augmenting stub resolvers with privacy features has gotten little attention in research. We assume that the general assumption is that augmenting stub resolvers will introduce performance overheads. We show that this assumption does not hold water. Our observations suggest that stub resolvers could play a vital role in safeguarding the user's privacy. We think, however, that providing the stubs with the right set of functions could play a significant role in protecting end-user privacy.

Our work shows an improvement of the current state of the DNS ecosystem where we resolve queries the old-fashioned way and add end-to-end encryption. Using an SoS resolver ensures not all query information is leaked to a single recursive resolver. Query information is not completely hidden, but ends up at TLDs and SLDs, typically ran by not-for-profit organizations.

Completely hiding all identifying information about an end-user is practically infeasible unless the user can trust some proxy to forward queries to name servers or recursive resolvers, without colluding with the recipient or eavesdropping the DNS packets. Hiding what the user is querying (Paged DNS, PIR) while preserving performance is not possible with current technology.

## REFERENCES


[1] Donald E. Eastlake 3rd. 1999. *Domain Name System Security Extensions.* RFC 2535. RFC Editor. http://www.rfc-editor.org/rfc/rfc2535.txt

[2] 4G.co.uk. [n.d.]. How fast is 4G? https://www.4g.co.uk/how-fast-is-4g/.

[3] Mark Allman. 2019. On Eliminating Root Nameservers from the DNS. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks (HotNets '19).* Association for Computing Machinery, New York, NY, USA, 1–8.

[4] Waqar Aqeel, Balakrishnan Chandrasekaran, Anja Feldmann, and Bruce M Maggs. 2020. On Landing and Internal Web Pages: The Strange Case of Jekyll and Hyde in Web Performance Measurement. In *Proceedings of the ACM Internet Measurement Conference.* 680–695.

[5] Waqar Aqeel, Balakrishnan Chandrasekaran, Bruce Maggs, and Anja Feldmann. 2020. On Landing and Internal Pages: The Strange Case of Jekyll and Hyde in Internet Measurement. In *Internet Measurement Conference.* ACM.

[6] Daniele E. Asoni, Samuel Hitz, and Adrian Perrig. 2018. A Paged Domain Name System for Query Privacy. In *Cryptology and Network Security*, Srdjan Capkun and Sherman S. M. Chow (Eds.). Springer International Publishing, Cham, 250–273.

[7] S. Bortzmeyer. 2015. *DNS Privacy Considerations.* RFC 7626. RFC Editor. http://www.rfc-editor.org/rfc/rfc7626.txt

[8] S. Bortzmeyer. 2016. *DNS Query Name Minimisation to Improve Privacy.* RFC 7816. RFC Editor. http://www.rfc-editor.org/rfc/rfc7816.txt

[9] Thomas Callahan, Mark Allman, and Michael Rabinovich. 2013. On modern DNS behavior and properties. *ACM SIGCOMM Computer Communication Review* 43, 3 (2013), 7–15.

[10] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. 1995. Private information retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science.* IEEE, 41–50.

[11] A. Cooper, H. Tschofenig, B. Aboba, J. Peterson, J. Morris, M. Hansen, and R. Smith. 2013. *Privacy Considerations for Internet Protocols.* RFC 6973. RFC Editor. http://www.rfc-editor.org/rfc/rfc6973.txt

[12] Peter B Danzig, Katia Obraczka, and Anant Kumar. 1992. An analysis of wide-area name server traffic: A study of the internet domain name system. In *Conference proceedings on Communications architectures & protocols.* 281–292.

[13] Wouter B de Vries, Quirin Scheitle, Moritz Müller, Willem Toorop, Ralph Dolmans, and Roland van Rijswijk-Deij. 2019. A First Look at QNAME Minimization in the Domain Name System. In *International Conference on Passive and Active Network Measurement.* Springer, 147–160.

[14] DNSCurve. [n.d.]. DNSCurve: Usable security for DNS. https://dnscurve.org/.

[15] Sebastian Egger, Peter Reichl, Tobias Hoßfeld, and Raimund Schatz. 2012. "Time is bandwidth"? Narrowing the gap between subjective time perception and Quality of Experience. In *2012 IEEE International Conference on Communications (ICC).* 1325–1330. https://doi.org/10.1109/ICC.2012.6363769

[16] Theresa Enghardt, Thomas Zinner, and Anja Feldmann. 2019. Web performance pitfalls. In *International Conference on Passive and Active Network Measurement.* Springer, 286–303.

[17] S. Farrell and H. Tschofenig. 2014. *Pervasive Monitoring Is an Attack.* BCP 188. RFC Editor. http://www.rfc-editor.org/rfc/rfc7258.txt

[18] Google. [n.d.]. Lighthouse. https://developers.google.com/web/tools/lighthouse.

[19] Google. 2019. Speed Index. https://web.dev/speed-index/.

[20] P. Hoffman and P. McManus. 2018. *DNS Queries over HTTPS (DoH).* RFC 8484. RFC Editor. http://www.rfc-editor.org/rfc/rfc8484.txt

[21] Austin Hounsel, Kevin Borgolte, Paul Schmitt, and Nick Feamster. 2020. D-DNS: Towards Re-Decentralizing the DNS.

[22] Austin Hounsel, Kevin Borgolte, Paul Schmitt, Jordan Holland, and Nick Feamster. 2020. Comparing the effects of dns, dot, and doh on web performance. In *Proceedings of The Web Conference 2020.* 562–572.

[23] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. 2016. *Specification for DNS over Transport Layer Security (TLS).* RFC 7858. RFC Editor. http://www.rfc-editor.org/rfc/rfc7858.txt

[24] Bert Hubert. 2019. Centralised DoH is bad for Privacy, in 2019 and beyond. https://labs.ripe.net/author/bert_hubert/centralised-doh-is-bad-for-privacy-in-2019-and-beyond/

[25] Eric Kinnear, Patrick McManus, Tommy Pauly, Tanya Verma, and Christopher A. Wood. 2021. *Oblivious DNS Over HTTPS.* Internet-Draft draft-pauly-dprive-oblivious-doh-06. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-pauly-dprive-oblivious-doh-06 Work in Progress.

[26] P. Mockapetris. 1987. *Domain names - concepts and facilities.* STD 13. RFC Editor. http://www.rfc-editor.org/rfc/rfc1034.txt

[27] P. Mockapetris. 1987. *Domain names - implementation and specification.* STD 13. RFC Editor. http://www.rfc-editor.org/rfc/rfc1035.txt

[28] David Naylor, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio Munafò, Konstantina Papagiannaki, and Peter Steenkiste. 2014. The cost of the" s" in https. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies.* 133–140.

[29] Erik Nygren. 2018. Architectural Paths for Evolving the DNS. https://blogs.akamai.com/2018/10/architectural-paths-for-evolving-the-dns.html.

[30] T. Reddy, D. Wing, and P. Patil. 2017. *DNS over Datagram Transport Layer Security (DTLS).* RFC 8094. RFC Editor. http://www.rfc-editor.org/rfc/rfc8094.txt

[31] Paul Schmitt, Anne Edmundson, Allison Mankin, and Nick Feamster. 2019. Oblivious DNS: Practical Privacy for DNS Queries: Published in PoPETS 2019. In *Proceedings of the Applied Networking Research Workshop (ANRW '19).* Association for Computing Machinery, New York, NY, USA, 17–19.

[32] Kyle Schomp, Mark Allman, and Michael Rabinovich. 2014. DNS Resolvers Considered Harmful. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks (HotNets-XIII).* ACM, New York, NY, USA, 16:1–16:7.

[33] Kyle Schomp, Michael Rabinovich, and Mark Allman. 2016. Towards a model of DNS client behavior. In *International Conference on Passive and Active Network Measurement.* Springer, 263–275.

[34] Sudheesh Singanamalla, Suphanat Chunhapanya, Marek Vavruša, Tanya Verma, Peter Wu, Marwan Fayed, Kurtis Heimerl, Nick Sullivan, and Christopher Wood. 2020. Oblivious DNS over HTTPS (ODoH): A Practical Privacy Enhancement to DNS. *arXiv preprint arXiv:2011.10121* (2020).

[35] T. Wicinski. 2021. *DNS Privacy Considerations.* RFC 9076. RFC Editor. http://www.rfc-editor.org/rfc/rfc9076.txt

[36] Peter Wu. 2019. The Cloudflare Blog: DNS Encryption Explained. https://blog.cloudflare.com/dns-encryption-explained/.