Technical Report
KN–2014–DiSy–004

Distributed System Laboratory

Stateless DNS

Daniel Kaiser, Matthias Fratz, Marcel Waldvogel,
Valentin Dietrich, Holger Strittmatter

Distributed Systems Laboratory

Department of Computer and Information Science

University of Konstanz – Germany

Abstract. Several network applications, like service discovery, file discovery in P2P networks, distributed hash tables, and distributed caches, use or would benefit from distributed key value stores. The Domain Name System (DNS) is a key value store which has a huge infrastructure and is accessible from almost everywhere.

Nevertheless storing information in this database makes it necessary to be authoritative for a domain or to be "registered" with a domain, e.g. via DynDNS, to be allowed to store and update resource records using `nsupdate` . Applications like the ones listed above would greatly benefit from a configurationless approach, giving users a much more convenient experience.

In this report we describe a technique we call Stateless DNS, which allows to store data in the cache of the local DNS server. It works without any infrastructure updates; it just needs our very simple, configurationless echo DNS server that can parse special queries containing information desired to be stored, process this information, and generate DNS answers in a way that the DNS cache that was asked the special query will store the desired information. Because all this happens in the authority zone of our echo DNS server, we do not cause cache poisoning. Our tests show that Stateless DNS works with a huge number of public DNS servers.

Table of Contents

1 Introduction

Stateless DNS is a technique that allows to use the existing DNS infrastructure as key value store. It does not need registration (like Dynamic DNS) and is restricted to zones shared by users of the same caching DNS servers. Instead of storing the information in authoritative servers, like usual DNS applications do, we use DNS caches for storage. To store information in a DNS cache,[1] hosts send a special "programming query" to our echo server, which uses only the information contained in this query to generate a DNS response the DNS cache will store. The programming query tells the echo server which label (key) the cache should use to store the information (value), allowing other hosts using the same DNS cache to retrieve the data with a DNS query for this label.

Our echo server does not need any configuration files or state; it consists of a single Perl script using the Perl DNS library,[2] making redundant or local deployment very easy. It is *not* necessary to have an echo server running within an institution. Any echo server instance can be used as long as it supports our query format. The lack of state also allows the use of anycast, improving fault tolerance.

To evaluate our technique we also implemented the echo server in Java and C++ and tested how well different methods allow us to store data in the cache of a huge list of public DNS caching servers.[3]

In this report we present

- an innovative way to use the existing DNS infrastructure,
- several methods to generate DNS answers that allow to store desired information in DNS caches,
- an evaluation of these methods, and
- an overview over possible applications.

[1] This can be the cache of a provider, of an institution, of a router at home, or the cache of any public DNS server.

[2] http://www.net-dns.org

[3] http://public-dns.tk/nameservers.xml

2 Functioning of Stateless DNS

Our stateless DNS technique allows any local[4] user to store DNS resource records in the cache of the local DNS server, thus allowing hosts who share the same caching DNS server to locate each other and offer services to each other. It works by sending a special *programming query* to the local DNS server. The authoritative server replying to this query is a special DNS server we call an *echo server*, which generates the answer solely from information contained in the query, without consulting a zone file or other database storing the contents of the zone. It is in a sense more stateless than other DNS servers, hence the name *stateless DNS*.

The generic form of a programming query is ***`data.method.alias.domain`***, where

`data` is the data we want to store,

`method` is the name of a method and determines the kind of answer the echo server will generate,

`alias` is an alias which can later be used to retrieve the data, and

`domain` is the authoritative domain of the echo server that will generate the answer.

Anyone using the same caching DNS server can later retrieve the stored data by querying for the alias, ie. for ***`alias.domain`***.

2.1 Basic Example

Let's assume the notebook in subnet 2 (Figure 1, $\boxed{1}$) sends a programming query by asking for an `A` resource record with the label

```
134.34.165.165.A.mynotebook.echo3.netfuture.ch.
```

to the local DNS server (Figure 1, $\boxed{2}$). Because our echo server is authoritative for this query, the local DNS server will send the query to this echo server (Figure 1, $\boxed{3}$). The echo server then generates a response from the information contained in the query. Because the query used the simple "A" method, the echo server will just answer with an `A` record:

```
mynotebook.echo3.netfuture.ch. IN A 134.34.165.165.
```

The local DNS server might now cache this `A` resource record, allowing e.g. the notebook in subnet 1 (Figure 1, $\boxed{5}$) to retrieve this resource record by asking for an `A` record with the label `mynotebook.echo3.netfuture.ch` (Figure 1, $\boxed{4}$).

This is a simple example to show the basics of how our stateless DNS technique works. Most DNS servers would not cache this A record. The reason behind this, and methods that works on almost all DNS servers, will be explained in the following sections.

[4] *local* in the sense that it is authorized to query the local DNS server

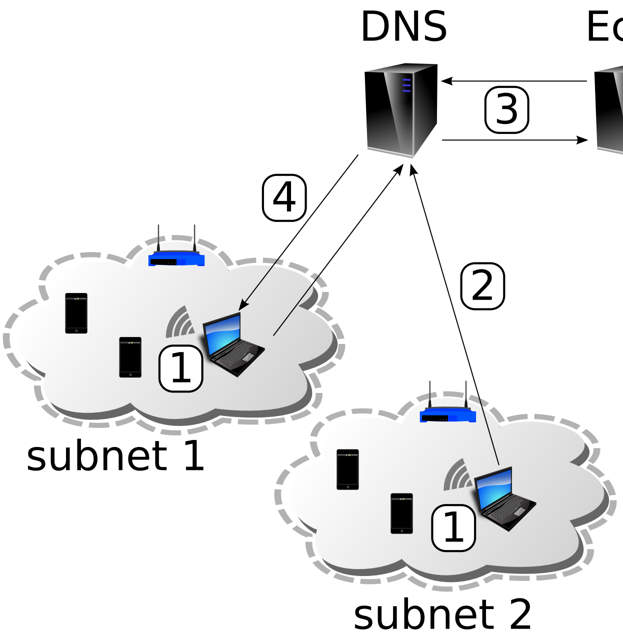

Fig. 1. Using our stateless DNS technique, the notebooks ($\boxed{1}$ and $\boxed{5}$) in subnet 1 and subnet 2 can exchange arbitrary DNS records, without any configuration or changes to the existing network infrastructure. This could be used e.g. so they can find each other and start direct communication, which otherwise isn't easily possible because they are in different subnets and thus in different multicast domains.

2.2 Bailiwick

To prevent DNS cache poisoning [1], many DNS servers implement Bailiwick rules. To make stateless DNS work, we have to heed these rules; if we don't, DNS caches will simply drop the resource records generated by our echo server.

The Bailiwick [1] is the domain of a DNS server which other DNS servers get, when they are referred to this server; thus the Bailiwick is dependent on and defined by other DNS servers. These servers use the Bailiwick of the server they were referred to, to check if they can trust the answers form that server.

It is important to only accept and cache answers that are in the Bailiwick of the queried referral server, meaning when asking for the `A` record of `bad.com`, the cache should not accept an A record mapping `good.com` to some IP address.

If e.g. a recursive DNS server gets the `A` resource record programming query

```
134.34.165.165.A.mynotebook.echo3.netfuture.ch A
```

it will eventually get a referral to `echo3.netfuture.ch`, which is handled by our echo server; thus the Bailiwick of our echo server is `echo3.netfuture.ch`, meaning caching DNS servers will only cache and accept resource records from our echo server if it is authoritative for those records.

On most DNS server implementations the Bailiwick rules are more strict. They do not just check if the answer is in the Bailiwick of the DNS server that gave the answer. Bailiwick rules are not specified in an RFC but [2] advises to only accept in domain records among other tips to make DNS more secure.

The Draft [3] advises among other tips to use the cache with care. It advises to only cache the first CNAME of a chain, to only use the first DNAME of a chain, and to use DNAMEs from cache only if that DNAME was not returned for a query. It also advises to do an authority query for NS after referral. Unbound[5] uses these strict rules.

Son et al. [1] give a very helpful analysis of Bind 9, Unbound and Mara DNS. In addition to that, our analyses of Bind 9 showed that its Bailiwick rules depend on whether dnssec-validation is activated. If it is, it just caches the first CNAME of a chain and asked again for this CNAME. It does not cache any further label the first CNAME points to. If dnssec-validation is deactivated, Bind stores a whole CNAME chain, but it still does not store any other type of record with the label the first CNAME points to.

2.3 Method Descriptions

Like stated above, our goal is to make a caching DNS server store a resource record generated by our echo server. Because there are different Bailiwick implementations, we developed different methods the echo server uses to answer a query. In the following sections we present these methods. We specify them by defining the query and the generated answer records.

Borrowing from BIND's zone file syntax, we use "@" as an abbreviation for the domain of the echo server; in our example it is echo3.netfuture.ch. For some of the methods we also need a special echo domain, which is 00.@, or 00.echo3.netfuture.ch in our example. Queries to this domain will always be answered in the following way:

```
Question: data.00.@
Answer:
  data.00.@ IN A data
```

Data before a further 00 will be ignored, such that *.00.data.00.@ is equivalent to data.00.@; this is required for some methods.

For each of the methods, we will present the example of trying to store certain data under the domain mynotebook.echo3.netfuture.ch in the cache of the local DNS server. We will present the answer our echo server generates for the particular example programming query and – for methods where it is not obvious – the query used to retrieve the data and the answer generated by this retrieving query.

In section 4 we show, for a huge number of public DNS servers, how well the different methods perform.

2.4 A: Direct Storage

The simplest class of methods are methods that tell the echo server to directly return the desired record. The A method simply returns the record we want the server to cache, even though the server did not ask for this record.

```
Question: data.A.alias.@ A
Answer:
  alias.@ IN A data
```

[5] http://unbound.net/

In most cases this method will not work, because the answer does not answer the question. But since the answer is within the Bailiwick of the echo server, it is not strictly incorrect to cache it, and some servers[6] do actually accept this simple method. It is *not* cache poisoning[7] because the echo server returning the answer is actually authoritative for this record.

For the example programming query

```
134.34.165.165.A.mynotebook.echo3.netfuture.ch A
```

our echo server will respond with[8]

```
;; QUESTION SECTION:
;134.34.165.165.A.mynotebook.echo3.netfuture.ch. IN A

;; ANSWER SECTION:
mynotebook.echo3.netfuture.ch. 10000 IN A 134.34.165.165
```

This method, as well as the following methods, cannot just be used to store `A` resource records; they can also be used for `TXT` and `SRV` records. Storing a `TXT` record works in the same way; we just have to ask for a `TXT` record in both the programming and the retrieving query, and provide appropriately formatted data. That is, for the example programming query

```
key=value.A.mynotebook.echo3.netfuture.ch TXT
```

our echo server will respond with

```
;; QUESTION SECTION:
;key=value.A.mynotebook.echo3.netfuture.ch. IN TXT

;; ANSWER SECTION:
mynotebook.echo3.netfuture.ch. 86400 IN TXT "key=value"
```

For a `SRV` record, an example query is

```
host.name-22.A.mynotebook.echo3.netfuture.ch SRV
```

our echo server will respond with

```
;; QUESTION SECTION:
;host.name-22.A.mynotebook.echo3.netfuture.ch. IN SRV

;; ANSWER SECTION:
mynotebook.echo3.netfuture.ch. 86400 IN SRV 0 1 22 host.name.
```

Note that even for `TXT` and `SRV` records, the method name is still `A` in this case, because it only serves to select the “Direct Storage” method. The record returned depends on the query type, which consequently has to be changed to `TXT` or `SRV`, respectively.

Since storing `TXT` and `SRV` records instead of `A` records works in the same way for all methods, we will not continue to mention this for the following methods.

[6] such as the DNS service included with Windows Server 2012

[7] At leas not in the classical sense.

[8] Using the syntax of the `dig` utility.

2.5 `C*`: CNAME Aliasing

Many DNS caches do not accept the `A` method because the label of the query does not match the label of the query. To circumvent this problem, we use a CNAME.

2.5.1 `CA` The `CA` method uses a CNAME that maps the label of the programming query (which was requested) to the alias under which we want to store the record, while also providing the data for this alias.

```
Question: data.CA.alias.@
Answer:
  data.A.alias.@ IN CNAME alias.@
  alias.@ IN A data
```

Using the `CA` method, for the programming query

```
134.34.165.165.CA.mynotebook.echo3.netfuture.ch A
```

our echo server will generate the following answer

```
;; QUESTION SECTION:
;134.34.165.165.CA.mynotebook.echo3.netfuture.ch. IN A

;; ANSWER SECTION:
134.34.165.165.ca.mynotebook.echo3.netfuture.ch. ↵
                  10000 IN CNAME mynotebook.echo3.netfuture.ch.
mynotebook.echo3.netfuture.ch. 10000 IN A 134.34.165.165
```

Since many DNS server implementations (see section 4) cache the `CNAME` but do not trust the `A` record, our echo server has a problem: The caching server will next ask for `alias.@`, the domain the `CNAME` points to, but the echo server cannot answer this query, because it does not have any state.

2.5.2 `CU` The `CU` method associates an arbitrary, non-resolvable CNAME with the alias. This can be used to efficiently store data, if the cache accepts this.

```
Question: data.CA.alias.@
Answer:
  data.A.alias.@ IN CNAME data.xy.
```

The `.xy` ccTLD was deliberately chosen in the "private use" `X*` range of the ISO 3166-1 alpha-2 codes to maximize the probability that it will continue to stay unassigned. This is explicitly *not* possible with a non-ccTLD top-level domain because of ICANN assigning generic TLDs. Also note that the `*U` methods, because they do not actually return any non-CNAME records, the type of query used for programming is immaterial.

When confronted with the example query

```
stateless.dns.CU.mynotebook.echo3.netfuture.ch A
```

our echo server will answer with a `CNAME` pointing nowhere:

```
;; QUESTION SECTION:
;stateless.dns.CU.mynotebook.echo3.netfuture.ch. IN A

;; ANSWER SECTION:
stateless.dns.CU.mynotebook.echo3.netfuture.ch. ↵
                              10000 IN CNAME stateless.dns.xy.
```

2.6 CC*: CNAME Chaining

If the CNAME method does not work, we need to make the echo server capable of answering for the label the CNAME points to. To achieve this without requiring the echo server hold state, we use the CNAME chaining methods. This exploits the fact that some servers cache the entire CNAME chain, and only ask again for the last label which actually resolves to the A record.

2.6.1 CCA Like the CA method, the CCA method returns a CNAME. But instead of pointing directly at alias.@ and giving an A record for alias.@, it returns a second CNAME which points to the echo domain 00.@ described above. This second CNAME contains in its label enough information to fully describe the desired resource record. This way, the server can safely query for the echo domain again and get the correct data, even though no state is held.

```
Question: data.CCA.alias.@
Answer:
  data.A.alias.@ IN CNAME alias.@
  alias.@ IN CNAME data.00.@
  data.00.@ IN A data
```

This way the echo server can answer if the caching name server asks for the second CNAME. The CCA method works with many DNS caches (section 4). Because they cache all CNAMEs in a chain, they will cache the first two CNAMEs, and just ask again for the label the last CNAME points to, which can be resolved by the echo server.

Sending the example query

```
134.34.165.165.cca.mynotebook.echo3.netfuture.ch A
```

to our echo server, it will answer in the following way.

```
;; QUESTION SECTION:
;134.34.165.165.cca.mynotebook.echo3.netfuture.ch. IN A

;; ANSWER SECTION:
134.34.165.165.cca.mynotebook.echo3.netfuture.ch. ↵
                 10000 IN CNAME mynotebook.echo3.netfuture.ch.
mynotebook.echo3.netfuture.ch. ↵
          10000 IN CNAME 134.34.165.165.00.echo3.netfuture.ch.
134.34.165.165.00.echo3.netfuture.ch. ↵
                                     10000 IN A 134.34.165.165
```

The retrieving query will yield the CNAME pointing to the echo domain 00.@, as well as the A record it points to.

```
;mynotebook.echo3.netfuture.ch. IN A

;; ANSWER SECTION:
mynotebook.echo3.netfuture.ch. ↩
           9985 IN CNAME 134.34.165.165.00.echo3.netfuture.ch.
134.34.165.165.00.echo3.netfuture.ch. 9985 IN A 134.34.165.165
```

2.6.2 `CCX` This method is like the `CCA`, but doesn't actually include the `A` record, thereby forcing the server to make another request for it. This comes from the observation that most servers make that request anyway, so that omitting the data can actually reduce network traffic. It is not expected to actually influence cacheability of the records.

```
Query: data.CCX.alias.@
Answer:
  data.A.alias.@ IN CNAME alias.@
  alias.@ IN CNAME data.00.@
```

2.6.3 `CCU` The `CCU` method combines the `CU` method with CNAME chaining. It is intended as a way to efficiently store data, as opposed to IP addresses.

```
Query: data.CCU.alias.@
Answer:
  data.A.alias.@ IN CNAME alias.@
  alias.@ IN CNAME data.xy
```

To the programming query

```
stateless-dns.ccu.mynotebook.echo3.netfuture.ch. IN A
```

our echo server answers:

```
;; QUESTION SECTION:
;stateless-dns.ccu.mynotebook.echo3.netfuture.ch. IN A

;; ANSWER SECTION:
stateless-dns.ccu.mynotebook.echo3.netfuture.ch. ↩
                 10000 IN CNAME mynotebook.echo3.netfuture.ch.
mynotebook.echo3.netfuture.ch. ↩
                              10000 IN CNAME stateless-dns.xy.
```

The retrieving query for `alias.@` will give the following answer, efficiently containing only the CNAME which contains the requested data.

```
;; QUESTION SECTION:
;mynotebook.echo3.netfuture.ch. IN A

;; ANSWER SECTION:
mynotebook.echo3.netfuture.ch. 9958 IN CNAME stateless-dns.xy.
```

2.7 D*: DNAME Aliasing

The class of DNAME methods uses DNAME records to map *all subdomains* of an alias to an "immutable" subdomain of the echo domain, utilizing the double-00 syntax 00.*.00.@ described above. Subdomains are then of the form *.00.*data*.00.@, so that the actual subdomain used for the retrieving query is ignored. These methods work very well on almost all DNS servers.

2.7.1 DA Establishes the desired alias as a DNAME to an "immutable" echo host under the echo domain, so that all *subdomains* should return the same data. We also pretend to follow the DNAME by including the nominal target host in the response.

```
Question: data.DA.alias.@
Answer:
  alias.@ IN DNAME 00.data.00.@
  data.DA.00.data.00.@ IN A data
```

In this response, the query label matches the answer label: Because the initial query is "affected" by the DNAME, the query label data.DA.alias.@ is mapped to data.DA.00.data.00.@. Since the next record is an A record for exactly this label, it also very likely that both answers will be cached. In case the caching server asks again for the A record, the echo server will be able to answer, because the data is included in the question as usual.

Because DNAMEs only affect the subdomains of a label, not the label itself, the retrieving query does not actually ask for the alias, alias.@, but for a arbitrary subdomain thereof, *.alias.@.

The programming query

```
134.34.165.165.da.mynotebook.echo3.netfuture.ch A
```

will make the echo server answer in the following way.

```
;; QUESTION SECTION:
;134.34.165.165.da.mynotebook.echo3.netfuture.ch. IN A

;; ANSWER SECTION:
mynotebook.echo3.netfuture.ch. ↵
       10000 IN DNAME 00.134.34.165.165.00.echo3.netfuture.ch.
134.34.165.165.da.00.134.34.165.165.00.echo3.netfuture.ch. ↵
                                     10000 IN A 134.34.165.165
```

The retrieving query

```
anything.mynotebook.echo3.netfuture.ch A
```

might cause the following answer

```
;; QUESTION SECTION:
;anything.mynotebook.echo3.netfuture.ch. IN A

;; ANSWER SECTION:
mynotebook.echo3.netfuture.ch. ↵
```

```
        9954 IN DNAME 00.134.34.165.165.00.echo3.netfuture.ch.
anything.mynotebook.echo3.netfuture.ch. 9954 IN CNAME ↵
             anything.00.134.34.165.165.00.echo3.netfuture.ch.
anything.00.134.34.165.165.00.echo3.netfuture.ch. ↵
                                     10000 IN A 134.34.165.165
```

In this case the CNAME was created by the caching name server. It is not necessary, but shows what the DNAME is actually doing to the query label. Like stated above, anything before the second 00 will be ignored. This is why

```
anything.00.134.34.165.165.00.echo3.netfuture.ch
```

will always return 134.34.165.165.

2.7.2 DX This method is identical to DA, but doesn't include the nominal target in the response. This should force the server to make another request, asking for data.DA.00.data.00.@.

```
Question: data.DX.alias.@
Answer:
  alias.@ IN DNAME 00.data.00.@
```

2.7.3 DU Similar to the CU and CCU methods, the DU can be used to store data in an efficient way.

```
Question: data.DU.alias.@
Answer:
  alias.@ IN DNAME 00.data.xy.
```

The desired alias is established as a DNAME to a host that doesn't resolve to anything, so that all subdomains should return a proper CNAME, but no queries should go to the echo domain. This CNAME contains the desired stored data. In this case, the 00 delimiter has to be interpreted by the retrieving host, because no echo server is available for the unresolvable domain.

Querying our echo server for

```
stateless-dns.du.mynotebook.echo3.netfuture.ch
```

will yield

```
;; QUESTION SECTION:
;stateless-dns.du.mynotebook.echo3.netfuture.ch. IN A

;; ANSWER SECTION:
mynotebook1.echo3.netfuture.ch. ↵
                           10000 IN DNAME 00.stateless-dns.xy.
stateless-dns.du.mynotebook.echo3.netfuture.ch. ↵
          10000 IN CNAME stateless-dns.du.00.stateless-dns.xy.
```

The retrieving query for

```
anything.mynotebook.echo3.netfuture.ch A
```

might yield the following result.

```
;; QUESTION SECTION:
;anything.mynotebook.echo3.netfuture.ch. IN A

;; ANSWER SECTION:
mynotebook.echo3.netfuture.ch. ↵
                              9905 IN DNAME 00.stateless-dns.xy.
anything.mynotebook.echo3.netfuture.ch. ↵
                     9905 IN CNAME anything.00.stateless-dns.xy.
```

Again the `CNAME` might be automatically created by the caching name server.

2.7.4 Semi-Atomic Retrieve-or-Store

One notable feature of the `D*` methods is that the retrieving query, `anything.alias.@`, is very similar in structure to the programming query, `data.DU.alias.@`. In fact, blindly sending the programming query yields a very desirable result: If the alias already exists in the cache, the stored value is returned. If not, the value included with the "blind" programming query is stored in the cache.

Consider the programming query

```
134.34.165.165.da.mynotebook.echo3.netfuture.ch A
```

If the DNAME for `mynotebook.echo3.netfuture.ch` is not yet in the cache, the caching server will consult our echo server, which will return a request as pictured under section 2.7.1. It also caches the following DNAME record:

```
mynotebook.echo3.netfuture.ch. ↵
        10000 IN DNAME 00.134.34.165.165.00.echo3.netfuture.ch.
```

Now suppose a different host decides to store the value `134.34.205.68` using the programming query

```
134.34.205.68.da.mynotebook.echo3.netfuture.ch A
```

Because there is already a cache entry for `mynotebook.echo3.netfuture.ch`, the caching server resolves the DNAME internally and returns

```
;; QUESTION SECTION:
;134.34.205.68.da.mynotebook.echo3.netfuture.ch. IN A

;; ANSWER SECTION:
mynotebook.echo3.netfuture.ch. ↵
               9954 IN DNAME 00.134.34.165.165.00.echo3.netfuture.ch.
134.34.205.68.da.mynotebook.echo3.netfuture.ch. 9954 IN CNAME ↵
            134.34.205.68.da.00.134.34.165.165.00.echo3.netfuture.ch.
134.34.205.68.da.00.134.34.165.165.00.echo3.netfuture.ch. ↵
                                            10000 IN A 134.34.165.165
```

The last label, `134.34.205.68.da.00.134.34.165.165.00.echo3.netfuture.ch`, is not in the cache, and will be sent to the echo server. Because it ignores everything before the left `.00.` delimiter, it will actually synthesize a record from the data portion of `134.34.165.165.00.echo3.netfuture.ch`, returning an `A` record of `134.34.165.165` – that is, the result of the first programming query.

Importantly, however, if the DNAME record had not been in the cache at the time of the second programming query, the caching DNS server would have found itself in the same situation as for the first programming query, and would have returned the data contained in the second programming query, an `A` record of `134.34.205.68`, while also caching

```
mynotebook.echo3.netfuture.ch. ↵
         10000 IN DNAME 00.134.34.205.68.00.echo3.netfuture.ch.
```

That is, it stores the data provided in the second programming query if and only if there is not yet an entry for the alias, `mynotebook.echo3.netfuture.ch`. It is interesting to note that this update happens as atomically as the cache updates of the caching DNS server. While this does not necessarily mean that it is atomic at all, it does appear suggestive for arbitration schemes or similar applications.

2.8 `N*`: Dynamic Delegation using NS Records

The class of `N*` methods uses `NS` records to delegate an alias. This leads to very powerful methods which aren't limited to simple data storage. Also, because delegations are at the core of the domain name system, these should be expected to be the most robust methods.

2.8.1 `NA` This method establishes a delegation for `alias.@`, by delegating to a server in the echo domain.

```
Question: data.NA.alias.@
Authority:
  alias.@ IN NS data.00.@
Additional:
  data.00.@ IN A data
```

Even if `alias.@` is delegated to an IP address that does not provide name server services, some caches store the IP address and allow the IP address to be retrieved later on. Since the name server is within the echo domain, our echo server can answer an `A` record query even if the additional section is discarded by the cache.

For the programming query

```
134.34.165.165.na.mynotebook.echo1.netfuture.ch A
```

our echo server will answer with the following record set.

```
;; QUESTION SECTION:
;134.34.165.165.na.mynotebook.echo1.netfuture.ch. IN A

;; AUTHORITY SECTION:
mynotebook.echo1.netfuture.ch. ↵
              86400 IN NS 134.34.165.165.00.echo1.netfuture.ch.

;; ADDITIONAL SECTION:
134.34.165.165.00.echo1.netfuture.ch. 86400 IN A 134.34.165.165
```

If the server receiving the delegation answers DNS queries, the alias and its subdomains become resolvable using this server. Otherwise, the data can sometimes be retrieved by explicitly querying for the nameserver with

```
mynotebook.echo1.netfuture.ch NS
```

The cache then returns

```
;; QUESTION SECTION:
;mynotebook.echo1.netfuture.ch. IN NS

;; AUTHORITY SECTION:
mynotebook.echo1.netfuture.ch. ↵
             86400 IN NS 134.34.165.165.00.echo1.netfuture.ch.

;; ADDITIONAL SECTION:
134.34.165.165.00.echo1.netfuture.ch. 86400 IN A 134.34.165.165
```

The additional section may not be present, so that either local parsing of the echo hostname or another query would be required.

2.8.2 NU

This method allows to use NS records for efficient data storage. It is similar to the other *U methods.

```
Question: data.NU.alias.@
Answer:
  alias.@ IN NS data.xy.
```

An example programming query for the NU method looks as follows.

```
stateless-dns.nu.mynotebook.echo1.netfuture.ch. A
```

Our echo server responds to it with:

```
;; QUESTION SECTION:
;stateless-dns.nu.mynotebook.echo1.netfuture.ch. IN A

;; AUTHORITY SECTION:
mynotebook.echo1.netfuture.ch. 86400 IN NS stateless-dns.xy.
```

In this case, the record always has to be retrieved with an NS query, ie. with

```
mynotebook.echo1.netfuture.ch NS
```

and will always contain an NS record only because the hostname is unresolvable:

```
;; QUESTION SECTION:
;stateless-dns.nu.mynotebook.echo1.netfuture.ch. IN NS

;; AUTHORITY SECTION:
mynotebook.echo1.netfuture.ch. 86396 IN NS stateless-dns.xy.
```

2.8.3 NR This method is like NA, but the name server is in its own Bailiwick.

```
Question: data.NA.alias.@
Authority:
  alias.@ IN NS nr.alias.@
Additional:
  nr.alias.@ IN A data
```

This is a very powerful method, because it allows to delegate a subdomain `alias.@` to a DNS server running on any machine, for the domain of the local DNS server. It only works if `data` contains the IP address of a host that is able to answer queries for `something.alias.@` in the following way:

```
Question: something.alias.@
Answer:
  alias.@ IN A IP
Authority:
  alias.@ IN NS nr.alias.@
Additional:
  nr.alias.@ IN A data
```

It has to return itself as the authoritative server for `alias.@` and can return an arbitrary IP address for anything in the `alias.@` zone.[9]

For test purposes, we implemented another stateless server, that fulfills these properties and returns `1.2.3.4` as IP address for all names under `alias.@`.

For the programming programming query

```
134.34.165.176.nr.mynotebook.echo1.netfuture.ch A
```

our echo server will answer in the following way.

```
;; QUESTION SECTION:
;134.34.165.176.nr.mynotebook.echo1.netfuture.ch. IN A

;; AUTHORITY SECTION:
mynotebook.echo1.netfuture.ch. ↵
                  86400 IN NS nr.mynotebook.echo1.netfuture.ch.

;; ADDITIONAL SECTION:
nr.mynotebook.echo1.netfuture.ch. 86400 IN A 134.34.165.176
```

Assuming the host with the IP address `134.34.165.176` runs the minimal server described above, the retrieving query

```
something.mynotebook.echo1.netfuture.ch
```

will retrieve the following.

[9] The use of `nr.alias.@` for the nameserver avoids complications if the caching DNS server decides to query the echo server again for the address of the host being delegated to. For `nr.alias.@`, the echo server replies with en empty record set, but not NXDOMAIN, indicating that `nr.alias.@` does exist. Because no address is provided, the caching DNS server should be more likely to reuse an existing address.

```
;; QUESTION SECTION:
;something.mynotebook.echo1.netfuture.ch. IN A

;; ANSWER SECTION:
something.mynotebook.echo1.netfuture.ch. 86400 IN A 1.2.3.4

;; AUTHORITY SECTION:
mynotebook.echo1.netfuture.ch. ↩
                 86088 IN NS nr.mynotebook.echo1.netfuture.ch.

;; ADDITIONAL SECTION:
nr.mynotebook.echo1.netfuture.ch. 86088 IN A 134.34.165.176
```

If there is no data, this yields a name server which is in its own Bailiwick, but has no address provided. For the resolver, this is an irrecoverable situation, but may end up forcing it to use an older address. This is especially important when the resolver queries again for the `AAAA` record: We don't provide an answer, but the resolver already has the A record anyway. Not repeating that A in the `ADDITIONAL` section is ugly, but we can't do that because we don't actually know that record.

It may be possible to update the host that is responsible for a certain alias by letting this host return a IP for the `nr.alias.@` host.

2.9 Short-Time Caching

A few DNS caches don't cache anything or have very strict Bailiwick rules, e.g. Unbound. For those rare cases we implemented a special echo server that caches the results for an alias until the first query for this alias (or until resources are exceeded), to be able to provide it to resolvers which ask again. Using this echo server, our technique works with almost all DNS implementations, but loses its advantage of being completely stateless.

3 Applications

In general, all applications that would benefit from a key-value store that is already widely distributed and accessible by anyone without registration, could use Stateless DNS. It is important to remember that only hosts using the same DNS cache can access the same database. For some applications, this kind of locality might be an advantage. The main application area for which we are currently using Stateless DNS is service discovery; we will also show other types of applications that benefit from Stateless DNS.

Since Stateless DNS does not work 100% reliable as of yet, it is best to have a fall back method to retrieve the data, e.g. in service discovery applications (section 3.1), or not being dependent on the key-value, e.g. when using Stateless DNS as a cache or index for a cache.

3.1 Service Discovery

Stateless DNS can be used to enhance service discovery. We use it to avoid multicast with both trusted and public hosts.

3.1.1 Service Discovery among Trusted Hosts In previous work [4,5], we presented privacy enhancing techniques for Multicast DNS Service Discovery (mDNS-SD) [6,7]. Instead of multicasting the service announcement and request messages for everyone to read, we send these messages directly – via unicast – to trusted "friends". To be able to offer configurationless service discovery over unicast, we need means to distribute the network parameters of a special socket, called *privacy socket*, when entering a network. To this end our privacy extension [5] uses a metaservice type, `_ppSOS`, whose service instance name allows finding friend's devices. This instance name is a random string that is exchanged once during an initial user pairing.

One way to distribute the *privacy socket* is to use Stateless DNS. The benefit of using Stateless DNS is avoiding multicast altogether, thus allowing service discovery in large institution's networks with several multicast zones. Our modified zeroconf daemon stores the `SRV`, `TXT` and `A` resource records, which are needed to resolve the `_ppSOS` service instance, using Stateless DNS, thus allowing hosts who share the same DNS cache server to bootstrap our privacy extension. This is done by sending a programming query with a label similar to the mDNS-SD domain of the offered `_ppSOS` service instance. In addition to the service type and instance name, it also contains the data of the service record that should be stored. The programming query we use to store the example A resource record containing the IP address `134.34.165.165` corresponding to the `_ppSOS` instance `prvGK8cApytjxKRd` has the following form:

```
134.34.165.165.A.prvGK8cApytjxKRd.echo.uni-konstanz.de
```

If, for example, the notebook in subnet 2 (Figure 1), uses this programming query, the notebook in subnet 1, which cannot receive the multicast announcements of the notebook in subnet 2, can now ask the local DNS server for the announced resource records asking for `prvGK8cApytjxKRd.echo.uni-konstanz.de`.

3.1.2 Public Service Discovery We also want to use Stateless DNS for public service discovery without any pairing requirement. To be able to still perform a service instance listing query, which mDNS-SD does by asking for `PTR` records via multicast, the first few hosts offering a service instance for a certain service type act as directory for all service instances of this type. This is done using the `NR` method. A host that wants to offer the service instance `myname` of type `_presence` will first ask for instances of this service type using the retrieving query

```
_presence_tcp.echo1.netfuture.ch. IN A.
```

(Note the lack of the separating `.` to keep the alias a single DNS label.) If there is no answer, it will establish itself as name server for this service type using the programming query

```
134.34.165.176.nr._presence_tcp.echo1.netfuture.ch
```

This will cause the echo server to generate the following answer

```
;; QUESTION SECTION:
;134.34.165.176.nr._presence_tcp.echo1.netfuture.ch. IN A

;; ANSWER SECTION:
134.34.165.176.nr._presence_tcp.echo1.netfuture.ch.
    86400 IN A 1.2.3.4

;; AUTHORITY SECTION:
_presence_tcp.echo1.netfuture.ch.
    86400 IN NS   nr._presence_tcp.echo1.netfuture.ch.

;; ADDITIONAL SECTION:
nr._presence_tcp.echo1.netfuture.ch. 86400 IN A 134.34.165.176
```

The host has now to answer for

```
myname_presence_tcp.echo1.netfuture.ch
```

with the data needed to connect to the service. All further hosts that want to offer a service instance of type `_presence` will contact this host using the NS record and transmit the data of their service instance to this host.

Users that want to discover a service instance of a certain type ask for the `NS` that is responsible for this service type and then ask the `NS` for a list of service instances of this type (service browsing phase); afterwards they can ask for the SRV and TXT records corresponding to desired service instance names (service resolution phase).

To mitigate name server churn, several hosts act as nameservers and we introduce a protocol to assign nameservers and a metaservice that allows name servers responsible for the same service type to synchronize data. We will describe this in future work.

3.2 Further Applications

We are working on further applications that benefit from stateless DNS. In this report we just want to briefly describe them.

3.2.1 P2P Networks Stateless DNS can be used to find nodes in P2P networks and to create overlay networks. This application is very similar to our service discovery application. It is possible to build overlay sub-networks for users of the same DNS cache which can later interconnect using other techniques. If nodes in P2P networks exchange some kind of pairing information, they can easily find each other using Stateless DNS even if their network parameters change.

3.2.2 Distributed Hashtable Stateless DNS could be used as index structure for distributed hashtables, as well as for re-joining of subnetworks that have become separated from each other. This works very similar to P2P applications of Stateless DNS.

3.2.3 Covert Communication Cunche et al. [8] propose a method for covert communication using Bit Torrent Trackers and distributed hashtables. Their method could also be adapted to using Stateless DNS.

4 Evaluation

4.1 Evaluation script

To evaluate how good the methods in section 3.3 work, we searched for public DNS servers and developed a testing script written in bash. For each server, the script first checks if the server is actually reachable and allows recursion for DNS lookup queries. Then, for each method, it generates a pseudo-random alias which is used in the following queries. After this, the basic scheme is as follows:

- Send a programming query to the corresponding server using the corresponding method (including the type of record asked in the query) and the generated alias
- Check if the record was cached by sending a query asking for the specified alias (and record type)
- Send a new programming query (updating query) using the same method and alias but different data
- Check if the cached record was replaced

4.1.1 Load Balancing Some server groups, like for example the Google DNS servers, are subject to load balancing. Therefore, it might happen that the programming query is handled by a different server than the retrieving query sent afterwards. In that case, the result would be negative even if the record was actually cached. Our approach to solution is to send multiple queries. In detail, if the result of the retrieving query is negative, we double the amount of programming queries sent and repeat the test, until the result is positive or a maximum number of queries is reached. If, at some point, the retrieval is successful, we first send more programming queries to decrease the probability that there still is some server in the load balancing group that did not receive a programming query. Thereafter a similar amount of updating queries is sent, with one checking query following.

With this approach of an exponential increase in the amount of programming queries, we preserve the quality of our results even in the case of load balancing, while also obtaining some information on the size of the load balancing groups. In some cases, if the number of servers in a load balancing group is much larger than the chosen maximum number of queries, it might still happen that we get a false negative result, but choosing a reasonable maximum number ensures that the number of such false negative results is very rare and does affect our overall results by only a negligible amount.

4.1.2 Retention Besides knowing if saving a record in the cache of a server works or not, there is another important information we are interested in: How long does the record *stay* in the cache. Therefore, as soon as we get a positive result for a server, a retention test subprocess is started. This process checks (in exponentially growing time intervals) if the record is still cached until the test fails.

4.1.3 Network problems Despite the initial test for reachability, some packets might be lost during the test, depending on how good the network connection is. To prevent a bad influence on our results, queries within the test are resent if they did result in a connection timeout.

4.2 Results

The results depend on which DNS server implementation is running on the corresponding server, therefore we grouped our results by the different DNS server implementations and analyzed each group with a non-negligible amount of representative public DNS servers. At our current state of research, this includes only the most widely spread implementation, namely the BIND9 server, but we plan to extent our analysis to other implementations like the ones described in section 4.2.2.

Version	a	ca	cca	ccx	ccu	da	dx	du	nr	na	nu	Overall
BIND9	0%	8%	90%	89%	89%	99%	99%	98%	99%	0%	0%	798/802
Overall	13%	31%	86%	85%	68%	66%	66%	62%	97%	0%	4%	2025/2062

Hiding Version Information As it is easy (and also recommended to prevent vulnerability attacks) to hide the name and version of the nameserver in use, most of our results correspond to servers where we could not find the DNS server implementation being used. Even so, those results give us some information about the general percentage of servers that will cache our record for one of the methods. Another observation is that a high amount of servers shows the same behavior as BIND 9, the easiest explanation being that they actually are BIND 9 servers with that information hidden.

4.2.1 BIND 9

- The Bailiwick rules prevent records to be cached if they do not match the query, thus the methods A and CA do not work.
- CNAME Chaining methods work as long as the server is not configured to check for DNSSEC, because only the first record of a CNAME chain has to match the original query.
- DNAME methods do work even with the DNSSEC check enabled, but the records cannot be updated.
- For records in the authority or additional section, BIND checks if the given NS record actually maps to a DNS server that is authoritative for the alias domain, thus the methods NA and NU do not work, but NR does (also regardless of the DNSSEC check).

4.2.2 Other Implementations Besides the BIND9 server, there are other nameserver implementations like Unbound, dnsmasq or Microsoft DNS. While we obtained some test results for these servers as well, the number of representatives is not large enough to be considered here. Even so, the overall results indicate that stateless DNS also works for most other implementations. Especially the NR method shows very good results for all servers tested, but also the DNAME methods appear to work in most cases, the only exception so far being Unbound, because it only takes a cached DNAME in consideration if the query returned this DNAME. Our echo server never returned the DNAME in conjunction with the retrieving query, which causes unbound to not apply the DNAME returned by the programming query to the answer of any retrieving queries.

5 Conclusion and Future Work

Stateless DNS is an easy to deploy and configurationless technique giving arbitrary network applications access to a key value store. It is especially useful to applications that do not need to use a general purpose key value store but want to store traditional DNS resource records, e.g. DNS service discovery. As our evaluation shows, our technique works very well with Bind 9, which is by far the most widely spread DNS server implementation.

In future work, we plan to

- give an analysis of the Bailiwick rules of other DNS server implementations,
- introduce new methods `NNA` and `NNR`, which allow to directly specify a secondary name server for delegation, and
- do a thorough evaluation of load balancing effects,
- elaborate the application we just briefly mentioned.
- try to mitigate some problems, e.g. `DNAME`s not working with Unbound, by signing the records using DNSSEC.

References

1. S. Son and V. Shmatikov, “The hitchhiker’s guide to dns cache poisoning,” in *Security and Privacy in Communication Networks*. Springer, 2010, pp. 466–483. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-16161-2_27 2.2
2. A. Hubert and R. van Mook, *Measures for Making DNS More Resilient against Forged Answers*, ser. Request for Comments. Internet Engineering Task Force (IETF), 2009, no. 5452. 2.2
3. W. Wijngaards, *Resolver Side Mitigations*, ser. Internet-Draft. Internet Engineering Task Force (IETF), February 2009. 2.2
4. D. Kaiser and M. Waldvogel, “Adding privacy to multicast DNS service discovery,” in *Proceedings of IEEE TrustCom 2014 (IEEE EFINS 2014 Workshop)*, 2014. 3.1.1
5. ——, “Efficient privacy preserving multicast DNS service discovery,” in *Workshop on Privacy-Preserving Cyberspace Safety and Security (IEEE CSS 2014)*, 2014. 3.1.1
6. S. Cheshire and M. Krochmal, *Multicast DNS*, ser. Request for Comments. Internet Engineering Task Force (IETF), 2013, no. 6762. 3.1.1
7. ——, *DNS-Based Service Discovery*, ser. Request for Comments. Internet Engineering Task Force (IETF), 2013, no. 6763. 3.1.1
8. M. Cunche, M. A. Kaafar, and R. Boreli, “Asynchronous covert communication using bittorrent trackers,” Tech. Rep. RR-8554, June 2014. [Online]. Available: https://hal.inria.fr/hal-01011739 3.2.3