Passive DNS Replication

Florian Weimer
fw@deneb.enyo.de

July 30, 2004

Abstract

This report describes a new approach to Domain Name Service (DNS) replication. The DNS system relies heavily on replication (based on zone file transfers) to achieve its reliability goals, but this form of replication typically requires cooperation from other DNS administrators. However, certain failures still occur in practice, and a decoupled, centralized replica of DNS data faces scalability issues if it is based on zone transfers.

Our proposed alternative, dubbed passive DNS replication, does not require cooperation from zone administrators and is able to recover from additional failures. It automatically adapts to real-world DNS query patterns, but it does not aim at complete replicas of individual DNS zones.

We also show other applications for the replicated data, some of them related to fighting worms, bots and other forms miscreant activity on the current Internet.

1 A quick introduction to DNS

In this section, we provide a very rough overview over the Domain Name System, viewing it as a distributed, replicated database. We omit a few details (such as different record classes and recent developments like DNS security).

1.1 DNS record types

DNS can be viewed as a distributed database that uses domain names as a key. Queries (also called *lookups* or *name resolution* attempts) specify a domain name and the record type to be returned. These queries are processed by *name services*, which answer them either by consulting local databases, or by querying other name servers. They return *resource records* (RRs, or records). These records include the domain name key, the record type, and the actual query result (which can be a domain name as well, or some other piece of data).

Common record types which are critical to Internet applications and DNS itself are:

- A records, which assign IP addresses to domain names,
- CNAME records, which map aliases to underlying domain names,
- MX records, which are used for mail routing,
- NS records, used by DNS itself to assign name servers to domain names, and
- PTR records, which can be used to provide a "reverse lookup" from IP addresses to domain names (in some cases).

Other record types are used in some cases (e. g. SRV records and TXT records), but at the moment, they are not critical for the operation of the Internet.

1.2 DNS organization and replication

The smallest unit of replication is a *zone*. Zones contain a set of records under a certain domain. In traditional DNS implementations, zones are replicated from a primary DNS server to secondary servers using various forms of *zone transfer* protocols (some of which support incremental transfers).

A zone can contain NS records that associate subdomains with other name services. This association is called *delegation*. A name server to which a zone has been delegated is said to be *authoritative* for this zone.

In the past, zone transfers used to be available to the general public. This is no longer the case, and it's also impractical for zones that contain millions of mostly unused records (according to DENIC eG (2004), there are 7.7 million domains registered directly under `.DE`, which translates to over 15 million resource records). Nowadays, zone file transfers are only allowed for designated secondary name servers (which provide backup replicas of the zone). In many cases, alternative protocols replace DNS zone file transfers.

1.3 Caching of stale data

DNS uses extensive caching based on a *time-to live* field. The TTL value is supplied by authoritative name servers the value configured by the zone administrator. Each second, non-authoritative servers decrease by one the TTL of the records they cache. If the TTL reaches zero, the record is expired from the cache.

As a result, DNS strongly favors data availability over data consistency. A zone administrator cannot invalidate already cached records. The TTL-based expiration is the only invalidation mechanism (besides manual intervention of cache administrators, of course). This means that most DNS data is potentially out-of-date or stale.

On the other hand, clients almost never query authoritative servers directly, but try to obtain DNS records from a previously configured fixed set of *caching name servers* (also called *caching resolvers*). This means that these servers can be used to alter the DNS view of many clients.

1.4 Query limitations

Basically, DNS only supports a single kind of query: supply a domain name and a record type, and receive the corresponding data. This means that before other keys can be used, they have to be rewritten into a domain name.

For example, *reverse lookups* submit an IP address and ask for a corresponding domain name. In this case, an IP address like `192.0.2.1` is encoded into the domain `1.0.2.192.in-addr.arpa`.

The necessary record types to support such queries have to be added by the zone administrator (in this case, of the `0.2.192.in-addr.arpa` zone or one of its parent zones). They are completely optional, and DNS does not enforce consistency with A records which map domains to the address `192.0.2.1`. As a result, it is often impossible to use PTR queries to obtain a reverse mapping from IP addresses to domain names.

Other potentially interesting queries are unsupported by DNS, either. It is not possible to query for domain names with certain prefixes or suffixes. For suffixes, it used to be possible to use zone file transfers, but for most zones (including critical top-level zones such as `.COM` or `.DE`), zone file transfers are no longer allowed. Prefix-based queries are totally unsupported, and so are a lot of other queries (for example "Give me a list of all CNAME aliases for this domain").

2 Failure scenarios

Passive DNS replication can mitigate some kinds of DNS failures. In this section, we briefly review some typical failure scenarios which have been relevant in the past. Unfortunately, typical name server operators are somewhat secretive about failures they experienced, and it is hard to categorize failures based solely on publicly accessible information.

2.1 Unreachable name servers

Nowadays, unreachable name servers are hardly a problem anymore. It's easy to set up zone transfers, and secondary DNS service is readily available, both commercially and as a mutual service within the Internet community. However, there are still zones whose importance is underestimated and for which no proper replicas exist (see Zone Labs, Inc. (2003) for a description of one such case).

Attacks on servers. Denial of service attacks have been attempted against DNS servers, but so far, even large, coordinated attacks could not affect the availability of important, widely-replicated zones such as the root zone (see Lemos (2002),

quoting Paul Vixie that "[t]here was never an end-user that said there was a problem").

2.2 Incorrect zone contents

A legitimate zone administrator might put incorrect data into a zone. Even if such mistakes are quickly corrected, it takes some time until the data expires from the caches (until the TTL reaches zero).

2.3 Replication of bad data

It is possible that the master copy is corrupted (for example, truncated), and that this copy is used to update some or all replicas, which become corrupted as well. Today, most large zone operators seem to have reasonable safeguards against the propagation of zone updates which remove far too many records, but such failures do still occur from time to time.

These errors are often reported as denial of service attacks because in many cases, implementation errors in widely deployed resolvers, operating systems and applications result in increased load on the affected authoritative name servers. When all name servers for a zone which contains frequently used resource records become unavailable for an extended period of time, it is not unusual to observe an increase in queries to these unreachable servers by one or two magnitudes. It can become a challenge to bring back a zone to operation after such a failure.

2.4 Manipulation of zone delegation

Manipulation of the parent zone (which contains a delegation to the interesting zone) can have a direct impact on the availability

Such attacks are relatively easy because these zones (such as the top-level zones like `.COM`, `.NET` and `.DE`) are maintained by using lots of different parties of varying trust. There are some safeguards, but they have failed in the past.

If such an attack is detected and it affects a high-profile zone, service providers sometimes restore the original delegation on their caching name servers, to restore the previous DNS view for their own customers.

Hijacking. In some cases, zone administrators, registries or registrars guard the zone data only very weakly against updates from unauthorized sources. As a result, an attacker can hijack a child zone by altering the delegation, pointing it to a server under his control (see Artmann (2000) for a report of a successful attack).

Such problems are not necessarily the fault of these zone administrators. The huge number of records in the widely used top-level zones mandates a high degree of automation. Furthermore, there has to be some process which allows domain owners to regain control of the domain even if they have lost the token they previously used to authenticate themselves with the zone administrator. Such recovery procedures must also work if the domain owner changed names (maybe as the result of a takeover). Even after a strong, positive authentication of a customer, there still remains a possibility that the customer's claim of domain ownership is no longer valid or has never been in the first place.

Domain hijacking cases are currently not a frequent issue. However, changes in the domain transfer process (see The Internet Corporation for Assigned Names and Numbers, 2004) might increase the number of successful attacks.

Deletion. As reported in Libbenga (2003), there are cases in which a registrar removes a delegation which is still active. Even if such steps are formally correct (following policies that were required by the US Government, see The Internet Corporation for Assigned Names and Numbers, 2003), they often result in severe disruption of service, and it takes some time before the error can be corrected.

2.5 Affects on other protocols

In general, DNS information is used by the client to obtain a host address which offers a particular service. Therefore, most of the time, one can simply use the IP address directly, even if name resolution is not possible on the current DNS and has been obtained by other means. However, there are a few exceptions.

HTTP. Name-based virtual hosts can be used to host a variety of sites on a single IP address. In recent versions of the HTTP protocol, the client

therefore transmits the domain name to which the request applies. Typical HTTP user agents (such as web browsers) do not provide any user interface to manually set the IP address, while keeping a certain domain name on the surface. This is not a real obstacle, but just a technical issue that prevents the easy application of out-of-band DNS data.

SMTP. For a long time, SMTP has been using DNS for routing of mail messages, that is, to determine the mail server (*MX host*) that handles mail for the domain part of an email address. This mail routing mainly occurs at the sending site. On the receiving site, mail is often not routed using the public DNS system. This means that it is possible to successfully send email even in case of a DNS failure.

However, anti-spam measures have greatly increased the dependency on DNS on the receiving side of an email message:

- Some mail servers only accept connections from IP addresses which have a valid PTR record.
- During the initial SMTP handshake, the connecting mail server (which acts as a client during the protocol) supplies a host name. Some servers apply extensive DNS-based sanity checks on the host name.
- A feature called *sender call-out verification* tries to start delivery of an error report back to the specified message sender. MX hosts which use this feature can only receive mail if they are able to send mail to the sender. (The idea is to suppress spam which carries completely forged sender addresses.)
- Some anti-spam applications use DNS as a distributed database to provide hints about the trustworthiness of certain IP addresses (*DNS blacklists* or DNSBLs). DNSBL lookup failures are typical not fatal to mail delivery, though.

Some of these kludges can filter out a lot of unwanted email messages without many false positives, but typical anti-spam configurations reject mail in case of DNS problems (be they global, or local to the sender or receiver), at least temporarily during the time the DNS problem persists. This effect cannot be mitigated at the sending side.

Additionally, typical mail user agents do not offer a reliable way to override mail routing (a problem similar to the HTTP case), but mail servers typically provide straightforward and very flexible means to control mail routing, overriding or supplementing routing information from DNS.

3 Applications of third-party DNS replication

If zone data is replicated and stored in a suitable form, historic information on DNS records is available, and queries which previously were not supported by the DNS system can be performed. This leads to a number of interesting applications.

3.1 Recovery of zone data

The first application of replicated data is recovery of the lost master copy. If the replica is decoupled from the master copy (and not completely overwritten by master copy updates), it is possible to compensate some of the failures mentioned in section 2.

If resource records are stored together with timestamp information, it is possible to recover the view of a zone at a certain date. In particular, incorrect records (maliciously or inadvertently) added later can be discarded.

This information can be used by Internet service providers to reconstruct zone data and locally hijack zones on their own caching name servers, restoring a previous, supposedly correct DNS view. As described in the previous section, overriding incorrect DNS information is often not an option for end users because most clients lack the necessary functionality (and users would have to resort to edit hosts files and enter hard-coded name resolutions, for example).

3.2 Malware containment

The major motivating factor for the development of passive DNS replication was the inadequacy of the PTR-based reverse lookup. Recall that DNS provides a means to map IP addresses back to domain names, but even though A records combine all the necessary data, PTR must be correctly con-

figured by DNS administrators, otherwise the A reverse lookup is impossible.

Malware often contains a hard-coded domain name which identifies a command and control host. The malware performs a lookup on this domain name to obtain a set of IP addresses, and contacts one of those servers. After that, it waits for incoming commands, and performs the requested actions (for example, scanning for more vulnerable hosts, or flooding a specified target with garbage packets).

Even if the malware is still operational on the victim's computer, some of its functionality is unavailable once the domain name has been removed from DNS. Therefore, knowledge of the domain name is important, otherwise it is impossible to contact a DNS administrator with a request for removal. (Naturally, an attacker never bothers to set up correct PTR records.)

The problem is that malware is typically detected *after* it has performed its domain name lookup. Even if it is possible to eavesdrop on the network traffic (which is technically infeasible in most service provider environments), the network traffic does not reveal the domain name. Only during reconnection after disruption or similar events, recovery of the domain name is possible. This adds a significant delay, which is sometimes unacceptable.

Reverse engineering of the malware binary can recover the domain name being used, but this task is time consuming and requires a copy of the malware in the first place, which is often impossible to obtain in a reasonable time frame.

If a replica of actually used domain names with their IP addresses is available, it is possible to store these records into a database and perform queries on this data set that are impossible on the traditional DNS system. In this case, one would index those records based on the IP address, which makes it very fast to obtain actually used domain names for individual IP addresses.

3.3 Blacklisting detection

As mentioned before, some anti-spam measures use DNS as a distributed database, e. g. to indicate that email that arrives from certain hosts should be rejected.

Some of those blacklists, blacklists, including a few which are used relatively widely, have a high false-positive rate because the DNSBL operators apply questionable heuristics or implement processes that try to maximize collateral damage. Sometimes, no proper notification of the operator of the host is attempted.

If DNS record data is available in form that allows arbitrary access patterns, it can be scanned for A records whose domains contain IP addresses which are encoded in the usual way (reversed dotted-quad notation followed by some domain indicating the blacklist, for example `1.0.2.192.dnsbl.example.org`), especially if the IP addresses belong to the service provider's own address range. If a match is found, an alert is triggered and a process which tries to resolve the issue is initiated.

3.4 MX theft and other policy violations

MX theft occurs when someone points an MX record to a loosely-configured foreign mail server, without proper authorization, and uses it as a backup mail relay for this domain.

While MX theft is not a real issue on the current Internet, other forms of policy violations are possible, especially on relatively open university or corporate networks. Additional web servers are installed, and domain names are pointed to them, without authorization from the responsible staff. Secondary name service for foreign domains is provided.

3.5 Replacement of documentation

Some network operators are required by law to provide a list of domain names used by them. In Germany, this mostly applies to networks in the public sector and is the result of an obscure combination of legal requirements and blanket authorization of certain government bodies.

Open university networks may have documentation for official domain names which have been centrally registered, but if anyone can run his or her own authoritative name server (both technically and in accordance with university policy), centralized documentation is very likely incomplete.

Replicas can provide a better approximation of the situation. It remains to be seen if the authorities accept them.

3.6 Detection of configuration errors

A wide range of DNS configuration errors can be detected quickly if zone data is available in a compact form. Common configuration errors include:

- A public A record references an IP address from private address space (which is not routed on the public Internet).
- Leaked local top-level zones, such as `.LOC` or `.LOCAL`.
- A CNAME record that in turn points to a CNAME record.
- MX records that point to CNAME records or hard-coded IP addresses.
- NS records that point to hard-coded IP addresses.
- Stale records of any kind.
- MX and NS records that point to hosts on which the corresponding services are firewalled.

Such configuration errors result in unnecessary delays, and erroneous CNAME records may even cause email messages to bounce with errors. Therefore, it is desirable to detect and correct them.

3.7 Trademark protection

In most jurisdictions, trademarks must be defended against (deliberate or accidental) infringement, otherwise they dilute and finally lose their status as trademark. DNS zone replicas can be examined for potential infringement.

In order to cut down the rate of false positives (e. g. domain names which are held by the trademark owner, but not used officially), the name of the name servers of those domains (as given in NS resource records) can be used. If the servers belong to the trademark owner, the company very likely also owns the domain. IP addresses can also be taken into consideration and compared to the address ranges normally used by the company.

This approach does not use any out-of-band data and is not affected by the poor data quality often found in those resources. For example, for some top-level zones, domain name WHOIS information is in a notoriously bad state and lots of entries are unmaintained or contain obviously forged data. Zone data, which is actually used for production purposes, is generally more correct and up-to-date, although it might lack details that (in some cases) are available in WHOIS registries.

Phishing. Much in the same way, some forms of "phishing attacks" can be detected. In these attacks, someone creates a web site which looks like the official site of the attacked company, under an official-looking domain name. The web site, completely operated by the attacker, collects personal information, such as account names and passwords. Later, the attacker uses the collected data to defraud the attacked company and its customers. Of course, the attacker does not have to use domain names which resemble official ones used by the company, and detection of the attack does not stop it. However, zone replicas can be used as a building block in a broader defense against such attacks.

3.8 DNS-related statistics

Replicated DNS data can be used for various statistics, provided that artifacts introduced by the method of replication are taken into account.

Interesting statistics are per-zone name server distribution (how many zones are served from multiple servers in different autonomous systems?), the number of domains served by different DNS providers, and the percentage of authoritative name servers which still offer zone file transfers for hosted zones.

4 Passive DNS replication

In this section, we discuss the key aspects of our DNS replication method, mostly independent of our implementation.

4.1 Why passive?

We call our replication method *passive* because the authoritative name servers (from which the data is replicated) are not actively involved. Rather, the method examines captured DNS response packets and extracts resource records from them. This leads to the following definition:

> *Passive DNS replication* is the process of capturing live DNS queries and/or responses, and using this data to build partial replicas of as many DNS zones as possible.

Specifically, we are not interested in traffic patterns (for example, query distribution over the day), or query statistics (how often a domain is requested, or who queries for which domains).

In practice, we concentrate on response packets because they contain all the data we need to construct a replica. Fig. 1 shows a typical DNS response packet. A query packet would only contain the fields up to the question section (and would contain literally the same data besides some flags). Looking at the answer, authoritative and additional section, it is possible to extract four resource records (the IP addresses of the web and name servers, and the two authoritative name servers for the zone `example.com`). It is also possible to infer the existence of a zone called `example.net` (or, strictly speaking, `ns.example.net`).

higher-level protocol header (e. g. UDP)
DNS header `QR`, `QCODE=0`, `RCODE=0`, `AA`
question section `www.example.com IN A`
answer section `www.example.com IN A 192.0.2.2`

authoritative section
`example.com IN NS ns.example.com`
`example.com IN NS ns.example.net`

additional section `ns.example.com IN A 192.0.2.1`

Figure 1: A DNS response packet

Compared to traditional replication of DNS data using zone file transfers, our method has two key advantages:

- Only those records which are actively used are replicated, greatly reducing the storage needs.
- No cooperation from zone administrators is required.

There first advantage is also a disadvantage. The replica will always be incomplete (and not up-to-date). On the other hand, zone file transfers result in very good coverage for a few zones, but overall coverage will still be poor because most name servers no longer allow zone file transfers. Lots of negotiations would be necessary, and we expect administrators of large and frequently used zones at large service providers to hesitate before they agree to replication. Even quite reasonable terms (such as those described in VeriSign, Inc., 2004) may hinder cooperation in research efforts and other activities.

4.2 Verification

Captured DNS responses are not necessarily legitimate. They could be completely forged, or they could contain additional, incorrect data. Name servers face similar problems, although they issue the queries themselves and have somewhat more control over the process. However, most spoofing vulnerabilities have been fixed that do not require a protocol change (the issue of a low-entropy query ID is still unresolved).

Three different operation modes are possible with respect to data verification:

- Responses are captured and analyzed. No verification is performed.
- Responses are correlated with queries, and only if a matching query has been seen before, a response is accepted. Non-authoritative answers and authoritative answers from name servers which are not, in fact, authoritative, are filtered out.
- The data part of responses is ignored, only the domain names and resource records are used as hints for independent DNS queries to standard caching name servers. The results of these queries are then recorded.

The first mode of operation has a very straightforward implementation. Bogus data can be injected easily, though.

The second approach is very complex and offers only a limited performance advantage over the third. It is still subject to some forms of spoofing (if queries are manipulated, too, and not just responses).

In the third case, loops must be avoided. The implementation is also somewhat more complex than in the first case (but it still much easier than on-the-fly verification because the verification process is delegated to off-the-shelf name server software). Large record sets can be handled correctly, too.

Verification itself can be complicated because the data that appears on the DNS system varies slightly from service provider to service provider, and a DNS response that cannot be verified on one network might be perfectly valid on another one. In some usage cases, one might even want to work with potentially spoofed data (because spoofed data might actually be used by clients as well, for example).

If the replicated data is mainly used to support queries that the DNS system cannot process directly, result sets based on replicated data can be verified against official DNS records at query time.

Therefore, we start by implementing the first approach. When the number of spoof records reaches unacceptable levels, we have to switch to the third option.

4.3 Truncated responses

If a query to an authoritative name server matches a large number of records (and the response UDP packet would exceed 512 bytes, the limit set by the DNS standards), some servers, instead of supplying only partial information, return an empty response with a set truncation (`TC`) bit. In response, the querying caching name server will issue a second query, this time over TCP.

Some forms of DNS query validation also work with truncated responses to trigger additional queries (see Pazi et al., 2003). This means that such behavior may even occur if the number of records is deliberately kept small, to avoid exceeding the 512 bytes limit.

Unfortunately, such truncated, empty responses cause problems for completely passive DNS replication, unless capturing TCP queries and responses (including some limited form of TCP stream reassembly) is implemented. This approach is very complex, therefore this problem should be addressed by punting these responses to an active verification process (third option in section 4.2).

4.4 Wildcard records

Wildcard records (and other forms of record synthesis by authoritative name servers) can lead to lots of responses with different records, but of little value to the applications mentioned above. Replication based on zone file transfers does not have this problem because wildcards are explicitly represented.

Currently, several top-level domains perform record synthesis. `.MUSEUM` has full wildcards, but is not widely used. The name servers for `.COM` and `.NET` answer with a synthesized A record if queried for a domain name that includes certain protocol violations (e. g. a label which contains non-ASCII bytes, with the most significant bit set).

The current level of record synthesis does not seem to cause any problems because the number of synthesized responses from top-level zones is fairly low. This might change once the replication effort extends to areas with non-Latin scripts. The kind of synthesis implemented by `.COM` and `.NET` could make a difference.

If the number of synthesized records reaches a level which is too costly, specific filters would have to be added to suppress these records.

4.5 Supporting zone recovery

As mentioned before, one application of an independent replica is recovery. This means that some form of historic data has to be provided. We could use database-specific means for this (for example, use a database which provides point in time recovery), or store reconstructed zone files in a revision control system. But the approach we chose is different. Based on the data extracted from DNS packets, we construct *DNS flows*. This name comes from a similar concept called network flow data which is used to reconstruct virtual connections on a packet-switched network such as the Internet (see Navarro et al. (2000) for a practical application of this method).

DNS flows map complete resource records (for example, A records consisting of a domain name and an IP address) to a time span. The time span corresponds to the interval of time during which the resource record has been observed.

In order to obtain the zone as it was seen at a particular point in time, the resource records for

the zone have to be collected (based on indexing or other means), and then the records whose first time-stamp is later than the specified point in time are discarded.

Based on the time the DNS flow ended, it is possible to implement an expiry mechanism, either at query time, or in the database itself, to reclaim storage.

5 The implementation

In this section, we describe a centralized implementation of passive DNS replication. Our implementation runs on UNIX-like systems, and uses `libenyo`, a C++ library that provides some reusable base components (including a DNS processing library).

5.1 Architecture

The implementation consists of four different active components which are arranged in a pipeline:

- sensors which capture name server responses and forward them to
- analyzers, which extract the data from those responses,
- a collector which consolidates data from various analyzers into a single database,
- and query processors which provide read access to the database.

This processing pipeline is shown in Fig 2. Data flow is from top to bottom. The analyzers write batches to a disk-based queue, and the collector communicates the query processors by a persistent database. All analyzers, the collector, and the query processors run on the same host.

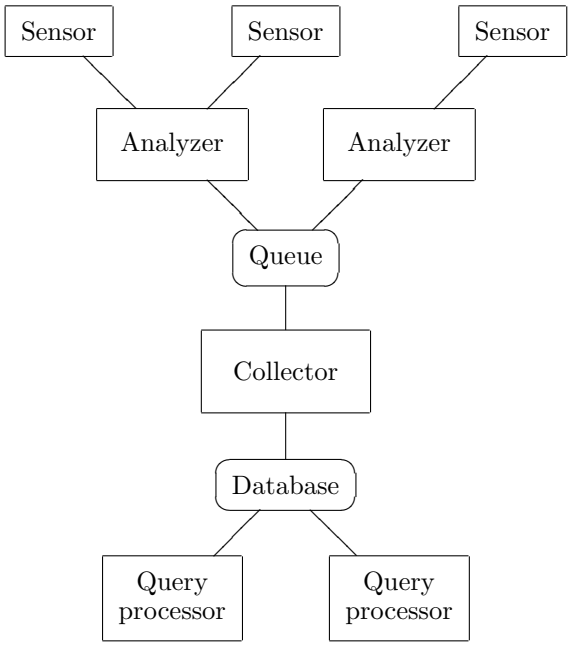

Figure 2: Architecture

5.2 Sensors and their placement

By design, the sensors are very light-weight. Their only task is to capture DNS packets sent by name servers. All further processing is deferred to the analyzers.

Currently, the sensor is implemented as a short Perl script which is built on top of the `Net::Pcap` Perl module. For each DNS packet, the script strips the link-layer, IP and UDP headers, prepends the magic string "`DNSXFR00`", and sends it over a standard UDP socket to an analyzer process.

Sensor placement. Sensors should be placed close to large, caching name servers, or at uplinks of networks containing caching name servers. Typically, a caching name server requests much more diverse data from other name servers than a single authoritative name server can provide. It makes sense to deploy sensors close to authoritative name servers only if above-than-average coverage of the zones served by those servers is desired.

Furthermore, as shown in Fig. 3, the sensor is best placed at a point where it does not catch traffic between the caching resolver and its clients. Here, the sensor captures packets sent to and from a caching name server on a monitor port of the DMZ switch. In such setups, we are not interested in DNS responses sent to the LAN because the resource records have already been recorded when the entry was cached for the first time.

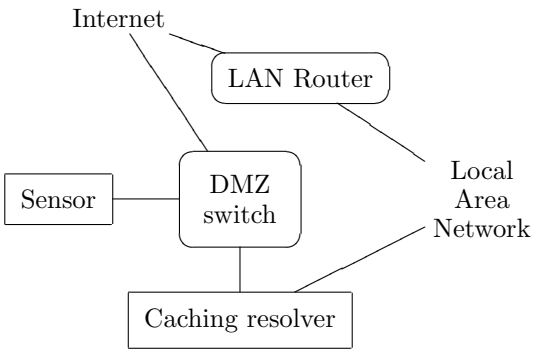

Figure 3: Sensor placement

Filtering using `libpcap`. We mentioned in section 4 that at this stage, we are interested in responses only and do not want to capture queries. Since the sensors use `libpcap`, it is possible to prefilter most of the network traffic with a filter expressions. Some possible choices are:

- "`udp and port 53`" forwards all DNS packets, responses with answer records, responses without them, and queries.
- "`udp and src port 53`" filters some queries, namely those that come from caching name servers (and clients) which do not use UDP source port 53.
- "`src port 53 and udp[10:2] & 0xf00f = 0x8000`" (the "`udp`" is implicit) only forwards DNS response packets which contain no error codes.
- "`src port 53 and udp[10:2] & 0xf00f = 0x8000 and udp[14:2] > 0`'' is even more restrictive. Only response packets are forwarded which contain an answer section which is not empty.

If the caching name server is not multi-homed (as in Fig. 3), a clause like "`and dst host 192.0.2.57`" can be added to the filter expression (in this example, `192.0.2.57` is the IP address of the name server). As a result, the responses sent to the client are filtered out.

Sensor performance. Although the sensors are implemented in Perl, performance is more than adequate even for large networks. All the filtering is performed by `libpcap` (or even the kernel, before it passes data to the library). This means that the number of packets which actually hits the Perl code is fairly low: on a large networks of over 10,000 hosts, we have observed 30 DNS packets per seconds with non-empty answer sections, and about the same amount with empty answer sections. The total number of DNS packets does not exceed 250 per second, even though some externally used caching name servers are on the network.

5.3 The analyzer

The analyzer can operate in two modes: In the first mode, it captures DNS packets on a specified interface (that is, it integrates a sensor component). In the second mode, it listens on a standard UDP socket for packets sent by one or more sensors. In both cases, the low-level protocol headers are removed. The DNS packet is parsed (using `libenyo`'s DNS implementation). The resource records found in the DNS packet (if any) are converted to textual form and written to the disk-based queue structure, in batch files which contain several thousand record each.

The queue uses an approach similar to Maildir (Bernstein, 2003) to allow for concurrent read and write operations. It also supports multiple readers which receive their own copy of the batches. The queue is also important for bridging database outages during software upgrades because it is continuously available (as long as the underlying UNIX file system is writable).

5.4 The collector

The collector reads analyzer output from the queue described above and updates a persistent database based on this information.

Database structure. Berkeley DB (Sleepycat Software, Inc., 2003) is used as an in-process database back end. Facilities of the `libenyo` library provide a modest level of separation between data structures and their representation in Berkeley DB tables (which is often a maintenance bottleneck

if non-SQL low-level database implementations like Berkeley DB are used).

Berkeley DB is used in transactional mode, so that query processes can access the same database while the collector is updating it.

Table 1 shows the database tables which are used by the collector. Indexes are available for all parts of the primary key (except the MX priority). In addition, the `domains` table is indexed by domain name and reversed domain name, and a partial index ins constructed from IP addresses which are embedded in the domain name (like `1.2.0.192.in-addr.arpa`).

The timestamps reflect the time span over which the record has been observed (first and last occurrence), implementing the DNS flows described in section 4.5.

Table	**Primary key**	**Data**
`domains`	domain ID (record number)	domain name
`A`	IP address, domain ID	time-stamp
`MX`	domain ID, MX host, priority	time-stamp
`CNAME`, `NS`, `PTR`	two domain IDs	time-stamp

Table 1: Collector tables

Operation. The collector process reads the files containing the batches, and removes them after successful processing. When a record is processed, its domain names are first converted to internal domain IDs (similar to interning symbols in Lisp). If the domain is seen for the first time, it is automatically inserted into the `domains` table, to assign a unique ID. Based on the IDs of the domains that are contained in a resource record, a key is constructed and it is used to locate the time-stamp information in the table corresponding to the record type. The time-stamp is updated and written back to the database if necessary. (If no record is found, one is created, with the appropriate time-stamp values.)

5.5 Query processors

The query processor implements two different interfaces: a command line interface and a FTP-style server mode which should be started from `inetd` to enable network access to the database. The main difference between both interfaces is the way query options are passed, query syntax and output format are the same.

Query types and query syntax. The query processes uses the indexes described in section 5.4 to provide the following query types:

- resource records for a domain (ordinary forward lookup, also available through DNS)
- reverse lookup, either by IP address (for A records) or by domain (for MX, CNAME, PTR and NS records)
- "blacklist" lookups (given `192.0.2.178`, it returns `178.2.0.192.dnsbl.example.net`)
- domain prefix lookup (list resource records for all domains beginning with a specified string of characters)
- domain suffix lookup (useful for recovering zones, together with point-in-time recovery)
- domain lookup based on IP addresses (for query an IPv4 prefix query, a list of records with domain names which contain a matching IP address is returned, e. g. `1.2.0.192.in-addr.arpa` matches a query for `192.0.2.0/24`)

The query syntax is very simple and uses a single string per query. An IPv4 address in dotted-quad notation triggers reverse lookup, and an IP prefix the domain lookup based on IP addresses. If the query string is neither an IP address nor an IP prefix, it is assumed to be a domain. If the string starts with "`.`", a domain suffix lookup is assumed. If it ends with "`.`", a prefix lookup is attempted.

The query processor attempts to provide a complete set of resource records that are related to the query in some way. For example, for an IP address, all A records referencing the IP address are returned. For each of the domains in the A records, the *CNAME closure* is returned. The CNAME closure for a domain is the smallest set such that it contains the domain, and for each CNAME alias which refers to a domain in the closure, it also

includes the alias name. In addition, an IP address lookup returns PTR records and blacklist entries. For domain lookups, both forward and reverse lookups are performed automatically.

Output format. The results are presented in table form, one resource record per row. Columns are separated by tabulators, rows by line feeds. The first column in a row is the domain name of a resource record, the second one the record type ("`A`", "`MX`" and so on), the third one the first data item (usually a domain name or an IP address). Further data items may appear in the following columns, depending on the record type.

This primitive output format is designed to fit in with the standard UNIX tool set (shell pipes, `grep`, `cut`, `sed`). This combination allows complex ad-hoc queries against the database (although manual query planning is required).

Point-in-time recovery. A special command line option can be used to limit the output to all that have been seen first *before* a certain time. This allows the user to exclude junk records which have been added to the zone at a later time.

Web-based front end. RUS-CERT offers a publicly accessible front end to their collector database at:

```
http://cert.uni-stuttgart.de/
stats/dns-replication.php
```

This public front end only supports a subset of the queries described above, and the number of records which are returned in response to a query is limited.

Requests for extended access to the database should be directed to the author.

6 Performance

Given that all the data is not mission-critical, strictly speaking, the collector uses Berkeley DB in a mode that preserves database integrity in case of application failures, but not system crashes.

The collector currently runs on a cheap, off-the-shelf Athlon x86 machine with only 512 MB of main memory. This machine has been upgraded with two SCSI disks (which can process about 200 transactions per second each).

Although the interning step was introduced to make the database more compact, the working set no longer fits into main memory. This is particularly noticeable during interning, which is bound by disk seek time. Time spent on interning although dominates the total processing time for updates.

Throughput ranges from 700 and 2,500 processed DNS response records per second, depending on the variability of the domains in those records. By increasing the batch size and sorting records before accessing the database to increase locality, throughput typically increases by a factor between two and three.

Table 2 shows the sizes of various tables. The indexes consume 2.6 GB of additional storage. Generally, they are slightly smaller than the corresponding tables, except for the three indexes on `domains`, which contribute another 1.8 GB in total.

Table	Records	Size
`domains`	24×10^6	777 MB
`A`	7.6×10^6	333 MB
`MX`	1.1×10^6	54 MB
`CNAME`	0.9×10^6	33 MB
`NS`	7.2×10^6	323 MB
`PTR`	8.3×10^6	262 MB
Total		1782 MB

Table 2: Collector table sizes

7 Coverage estimate

In this section, we try to estimate the coverage of the DNS replication server running at RUS-CERT.

Table 3 shows data for five different hosts at a German web hosting company. The first column lists the number of domains hosted on each server, the second indicates the number of sites (there are some sites without any domain), the third the sites which are actually active (having at least 1,000 hits in the last month). The fourth column shows how many domain names have been replicated. The final columns are coverage estimates, the ratio of replicated domains to hosted domains, and the ratio of replicated domains to active domains. The number of active domains is estimated by multiplying the total number of domains by the quotient of active sites and total number of sites.

# Dom	Sites	Active	Seen	D%	A%
15382	7423	2089	248	1.6	5.7
12716	12805	1272	67	0.5	5.3
21357	8337	1854	10	0.0	0.2
9498	10935	1406	109	1.1	8.9
10320	10449	1617	118	1.1	7.4
69273	49949	8238	552	0.8	4.8

Table 3: Coverage estimate

The results, while not very encouraging, suggest that the current replication system (with sensors based in Germany) catches between 0.8 and 4.8 percent of all German domain names used by active web sites.

8 Privacy Implications

Two different kinds of personally identifiable information arises in the context of domain names: data on the circumstances of the query (IP addresses involved, time of query, the domain name requested), and the actual contents of a domain name.

The first issues are easily addressed by the placement of the sensors. If the guidelines outlined in section 5.2 are followed, the sensor only observes inter-server traffic. This means that end user IP addresses cannot be recovered at that point. Thanks to caching, most queries do not result in inter-server queries, which further reduces the potential for misuse. Using flow data (see Navarro et al., 2000), it might be possible to correlate the time-stamp information and server IP address information in both data sets and thus obtain the end user addresses for some non-cached queries, but this privacy invasion is more a result of flow logging than of our DNS replication effort.

The second set of issues is harder to dismiss. There are a few web service providers which use wildcard A records and session IDs embedded into domain names to implement user tracking. In one case, these domain names are embedded in "web bugs" and are used to track users without their consent, so the publication of these domain names does not do any harm (because it dilutes the user identification, which is increasing privacy in this case). In other cases, the domain name contains a session ID within a web application.

As a consequence, the publicly accessible database front ends delay the availability of new resource records by 20 minutes. It is expected that after this time period, potentially affected sessions have expired.

9 Conclusion and future work

In this report, we motivated the need for passive DNS replication, described an architecture and an implementation, and presented some statistics about the DNS replication service running at RUS-CERT. This service is already being used for CERT-related activities, configuration checking, and policy enforcement by network administrators.

Of course, we plan to extend the sensor network to increase coverage, but we are going to address some of the shortcomings identified above as well, as described below.

More sensors. Beyond highly frequented web sites and mail servers, DNS queries often target domains of regional interest. This means that more sensors are needed to reach reasonable coverage at a global level. Offers to host sensors should be directed to the author.

Verification. At one point, it will be necessary to implement some kind of verification (section 4.2). At first, only a subset of all responses will be subject to verification (to tackle the truncation problem described in section 4.3).

Advanced queries. Currently, our query processing tool is quite limited. While it has been demonstrated that the current query tools can be integrated into shell scripts to implemented more advanced queries, tighter integration with other data sets (such as RIR database information and the Internet routing table) is desirable.

Support for regular expressions is also an interesting option, but it remains to be seen if it can be implemented efficiently on millions of domains.

Meta-replication. Growing interest in passive DNS replication will result in the need to replicate the replica (for example, for research purposes at

other sites). The time-stamp information (which is part of the DNS flows) is certainly useful for incremental replication, but the details still have to be worked out.

Feasibility of some applications. The feasibility of some of the applications presented in section 3 still has to be demonstrated in practice. In particular, it would be very interesting to try if passive DNS replication can be used to detect some kinds of phishing attacks.

References

DENIC eG, *Statistics* (2004), URL `http://www.denic.de/en/domains/statistiken/index.html`.

Zone Labs, Inc., *Temporary loss of transatlantic connectivity affected some ISPs, ZoneAlarm users* (2003), URL `http://zonealarm.com/store/content/company/aboutUs/pressroom/pressReleases/2003/outageFAQ.jsp`.

R. Lemos, *Assault on Net servers fails* (2002), URL `http://zdnet.com.com/2100-1105-963005.html`.

P. Artmann, *Freemail-Provider GMX blockiert* (2000), URL `http://www.heise.de/newsticker/meldung/8640`.

The Internet Corporation for Assigned Names and Numbers, *Policy on transfer of registrations between registrars* (2004), URL `http://www.icann.org/transfers/policy-12jul04.htm`.

J. Libbenga, *Why Spamcop got yanked* (2003), URL `http://www.theregister.co.uk/2003/11/03/why_spamcop_got_yanked/`.

The Internet Corporation for Assigned Names and Numbers, *Amendment 6 to ICANN/DOC Memorandum of Understanding* (2003), URL `http://www.icann.org/general/amend6-jpamou-17sep03.htm`.

VeriSign, Inc., *TLD zone file access program* (2004), URL `http://www.verisign.com/nds/naming/tld/`.

G. Pazi, D. Touitou, A. Golan, and Y. Afek, United States Patent Application 20030070 A1, United States Patent Office (2003).

J.-P. Navarro, B. Nickless, and L. Winkler (2000), URL `http://www.usenix.org/publications/library/proceedings/lisa2000/full_papers/navarro/navarro_html/`.

D. J. Bernstein, *Using maildir format* (2003), URL `http://cr.yp.to/proto/maildir.html`.

Sleepycat Software, Inc., *Berkeley DB* (2003), URL `http://www.sleepycat.com/docs/`.

Acknowledgments

The author wishes to thank Karl Gaissmaier for helpful suggestions regarding `libpcap` filter expressions.

RUS-CERT kindly provides resources to run the implementation of passive DNS replication described in this report. Several third parties donate DNS response data, making this a distributed effort by the Internet community.