Modern DNS as a Coherent Dynamic Universal Database

Paul Vixie <vixie@isc.org>
Akira Kato <kato@wide.ad.jp>

June, 2004

Abstract

The Domain Name System (DNS) allows zone data to change dynamically based on secure protocol messages, and the changed data can be globally visible after only a short delay. This report explores these features of DNS, including server and client implementation issues, and shows an example of their use in creating an automated Real Time Blackhole List based on reception of viruses from infected remote computers.

1. Introduction

DNS is a coherent autonomous distributed hierarchical database, designed in 1987 [RFC1034] to map Internet host names to their Internet Protocol (IP) addresses and to replace the ubiquitous `HOSTS.TXT` file. Information was entered into DNS using out of band mechanisms such as flat text files edited or generated by domain administrators, and was expected to be changed only rarely, on the order of hours or months.

After 17 years of evolution, the definition and scope of DNS has grown. New features to be discussed in this paper include the following:

- New information can now enter DNS dynamically, using secured protocol messages [RFC2136],[RFC2845]
- It is no longer necessary to periodically poll for DNS zone changes, greatly reducing propagation delay [RFC1996]
- Changes to DNS zone data can be transferred incrementally, allowing large zones to change quickly and often [RFC1995]

These new features make it possible to deliver services through DNS that were not originally possible. One example of a coherent dynamic universal database is a shared global real time blackhole list (RTBL) for e-mail.

2. DNS Changes

While a fully detailed explaination of dynamic update [RFC2136], real time change notification [RFC1996], incremental zone transfer [RFC1995], and transaction signatures [RFC2845] would merely repeat the Internet RFC specifications on these topics, it is useful to revisit these four protocol changes in overview.

Given the perspective allowed for by several years of implementation and deployment experience, here is what we now know:

2.1. Dynamic Update

In traditional DNS, all zone changes were made out of band, usually to a so-called *master file* which was either machine-generated or human-edited. The [RFC2136] proposal for in-band dynamic update allows zone changes to occur as a result of DNS protocol messages. These messages can come from DHCP agents, management consoles, or any other manual or automated method. Updates are atomic, reliable, and can be declared to be idempotent. Update authentication can be by TCP/IP source address, or by transaction signature (as specified in [RFC2845] and described in the following subsection.)

Updates are applied to the zone at the master server, which is the apex of a distribution graph containing slave servers, caching servers, and clients. If the master and slave servers for a zone all implement real time zone change notification (per [RFC1996]) and incremental zone transfer (per [RFC1995]) then the result will be nearly instantaneous zone changes visible at all of a zone's authority servers starting from in-band DNS protocol messages specifying dynamic updates. If a moderately low negative time-to-live (TTL) is specified as the zone's *SOA minimum* (start of authority record's *minimum* field), and if an equally low TTL is specified as the zone's default TTL, then caching servers and clients will see zone changes in close to real time.

2.2. Real Time Change Notification

Another traditional constraint in DNS was that zone changes could only occur at predefined intervals (called *SOA refresh* and *SOA retry*). Zone slave servers would only contact the zone master server every *SOA refresh* seconds (or *SOA retry* if there had been a prior communications failure) to check for an updated zone serial number. All changes to the zone that occurred within this interval were published simultaneously, as a batch. This led to either unfortunately short polling intervals, or unfortunately long embargoes on additions, changes, and deletions to zone data.

In [RFC1996], a method was proposed for zone master servers to signal to published and/or known zone slave servers the existence of a new zone, by transmitting only the zone's new serial number. If this serial number differs from the one the slave server is currently using, then an early *refresh* operation is triggered, whereby each participating zone slave server can verify the zone's serial number, and initiate a zone transfer if appropriate.

Wide scale deployment of [RFC1996] has been an unqualified success, and it is safe to say that no zone of any importance is served using software that does not comply with the [RFC1996] recommendation. This means most zone changes are signalled to slave servers within a few seconds of a zone change on the master server, and if incremental zone transfer is also in use, then the zone changes will be propagated within the same short period.

It's necessary to distinguish between the availability of new zone content on the authority servers (master and slaves), as compared to the definite use of changed data, which is not guaranteed until after the time-to-live (TTL) of the old data has expired. Note that for previously nonexistent data, there is an *SOA minimum* that sets the maximum interval for so-called "negative caching." The net effect of DNS caching is that changing the zone content on the master and slaves is useful but not a cure-all for staleness, due to positive and negative caching. (See Section 3.3.1 for more information on this topic.)

2.3. Incremental Zone Transfer

The traditional model for synchronizing content between zone master servers and zone slave servers requires that a complete copy of the zone content be transmitted from the master server to all slaves whenever this content is changed. For zones of nontrivial size or rate of change, this "full copy" model is prohibitively expensive. Historically, this meant that large zones could only be changed one per day or even once per week, and correspondingly required data which had to change on a more frequent basis to be placed in small separate zones that could be transferred frequently without high cost. These constraints were unfortunate, and acted against correctness in overall system designs involving DNS.

In [RFC1995], a method was proposed for transferring only incremental changes from the zone master to each zone slave. These changes, even if very frequent, were small enough to be transmitted and stored at only nominal cost. After subsequent wide scale adoption of that proposal, it is now common for zones of arbitrary size to change their content many times per day, sometimes even many times per minute. The [RFC1995] proposal is now universally considered a success, and is an Internet Standard.

Early software implementations of [RFC1995] (such as ISC BIND8 and early ISC BIND9 [ISCBIND], Microsoft DNS, and others) were constrained to only be able to transfer incremental zone changes if the source of those changes was a DNS Dynamic Update [RFC2136]. As of ISC BIND9.3, any zone change including complete regeneration or human editing can still be transferred as a minimal incremental change. From this we conclude that [RFC1995] has matured.

2.4. Transaction Signatures

During the preparation of the dynamic update proposal (see [RFC2136]), it became necessary to be able to authenticate a transaction's initiator. A general mechanism was proposed, called TSIG ([RFC2845]), by which any cooperating initiator and responder could share secrets offline or out of band, and use these shared secrets to prove the identity of a transaction's initiator and responder. While the out of band nature of these shared secrets is a harsh scaling limit, TSIG has shown its value both in enterprise contexts (where all initiators and responders are under the same administration) and in zone hosting contexts (where a zone's master and slaves are operated by cooperating parties, who already have an out of band administrative channel.)

Proposals to authenticate DNS transactions without out of band shared secrets include GSS-TSIG [RFC3645] and SIG(0) [RFC2931], which are beyond the scope of this paper.

2.5. Summary of Protocol Changes

IETF's "DNS I-N-D" project which included incremental zone transfers ("I"), real time zone change notification ("N"), and dynamic zone updates ("D") has remade DNS from its traditional static model into a modern and robust real time global publication medium for any DNS-compatible data.

With these features now available, it has become possible to deploy services using DNS rather than constructing a new coherent autonomous distributed hierarchical database for every new service or application.

3. DNS Services Example – Dynamic RTBL

Malfeasants now routinely employ viruses and worms in order to organize large numbers of always-online home computers into an unwitting infrastructure for launching spam, distributed denial of service, and other attacks. This author has observed that more than half of all host computers who attempt to deliver worms or viruses will also attempt to deliver spam within the following four weeks.

Given this correlation, it would be valuable to maintain a searchable database of "hosts who have recently tried to send us a virus," so that subsequent e-mail launched from these hosts can be rejected without question. (This is useful even in the case of e-mail worms, since if a host is willing to send a worm, it's a safe bet that anything else offered later on by the same host can be safely rejected or dropped.)

Given the wide spread adoption of DNS as a bearer of "real time blackhole list" (RTBL) data [MAPSRBL], the problem can be reduced to:

> *how can programs on our mail and other servers securely update a DNS zone used as a "blackhole list" and then grant all of our servers fast and coherent access to this zone?*

The full data flow necessary for this application is shown in Figure 1.

3.1. Software and Hardware Selection

The DNS server and client software used in this example was [ISCBIND] (version 9.3.0). The operating system was FreeBSD 5.2.1/AMD64.

3.2. Implementation Details

Implementation consisted of several small BSD UNIX shell-level utilities and some configuration and setup work for ISC BIND9 and the Postfix mailer [POSTFIX].

3.2.1. Adding Host Addresses – addhost.sh

```
#!/bin/sh

node=‘echo $1 | awk -F. \
      ’{print $4 "." $3 "." $2 "." $1}’ \
     ‘; shift
zone="ra.vix.com"
server="ns.vix.com"
ttl="1800"
kf="/var/ra/Kupdate-ra.+157+43810.key"

( echo server $server
  echo zone $zone
  echo prereq nxdomain $node.$zone
  echo update add $node.$zone $ttl \
    A   0.0.0.0
  echo update add $node.$zone $ttl \
    TXT created ‘date +%Y%m%d%H%M%S‘
  if [ $# -gt 0 ]; then
    echo update add $node.$zone $ttl \
      TXT reason $@
  fi
  echo send
) | nsupdate -k $kf /dev/stdin

exit $?
```

This utility uses a shared key file (given as the `nsupdate -k` argument) to authenticate a dynamic update to the `ra.vix.com` zone. Command line arguments to this utility are as follows:

$1	host address
$2..N	reason for entry

This utility adds one A RR (address record) and one or more TXT RRs (text records) at the name corresponding to the reversed (in-addr) form of the host address. For example, if host 12.10.104.137 were added on 2004-05-28 at 14:12:24 (UTC) then the result would look something like this:

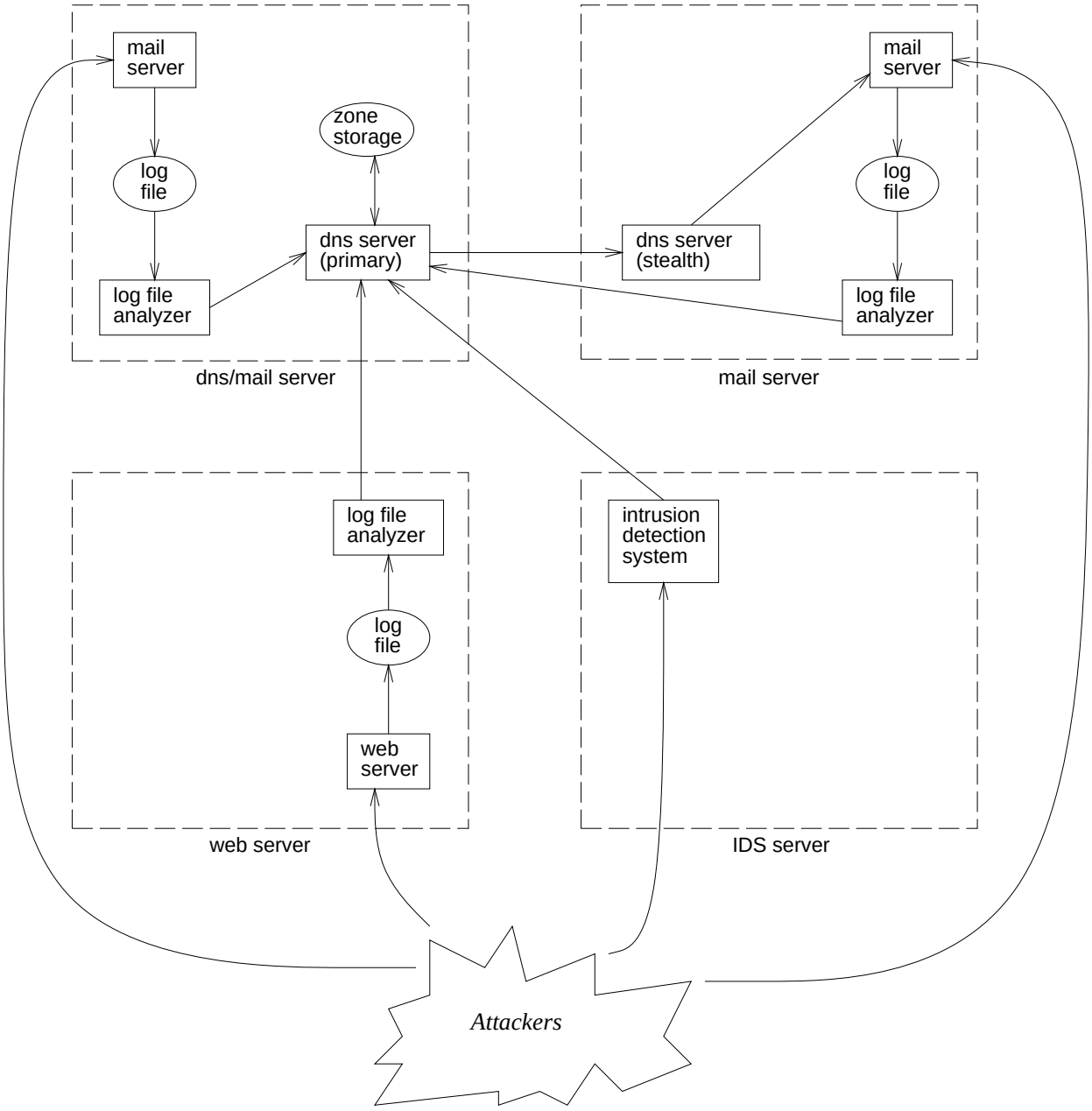

Figure 1. RTBL Data Flow

```
$ORIGIN 104.10.12.ra.vix.com.
137 A   0.0.0.0
    TXT "created" "20040528141224"
```

Note that a *prerequisite* for this update is the nonexistence of any records at the domain name corresponding to this address. If this condition is not met, then the update will not be applied.

For this "blackhole list," only the existence of an A RR (address record) matters, and the exact address does not matter. Thus `0.0.0.0` is used. The TXT RR (text record) shown is used for expiry (see Section 3.2.3).

3.2.2. Deleting Host Addresses – delhost.sh

```
#!/bin/sh

zone="ra.vix.com"
server="ns.vix.com"
ttl="1800"
kf="/var/ra/Kupdate-ra.+157+43810.key"

for N
do
  node=‘
    echo $N |
    awk -F.
      ’{print $4 "." $3 "." $2 "." $1}’
  ‘
  echo server $server
  echo zone $zone
  echo prereq yxrrset $node.$zone a
  echo prereq yxrrset $node.$zone txt
  echo update delete $node.$zone
  echo send
done | nsupdate -k $kf -t 600 /dev/stdin

exit $?
```

This utility's arguments are a list of host addresses whose information is to be removed from the private blackhole list. Each one is converted to "in-addr" format and sent to the `nsupdate` utility with *prerequisite*s of existence of A (address) and TXT (text) records, and a request for deletion of all records at that name.

3.2.3. Expiring Host Addresses – expire.pl

This PERL utility is not reproduced here. Its purpose is to output a list of hosts from our private blackhole list that were added more than 45 days ago. These are then sent to the `delhost.sh` utility for deletion.

The design of this utility is simple. During a zone transfer (`AXFR`) of the zone, matched sets of A (address) and TXT (text) records are found, and if a `created` date is available and if that date is more than 45 days in the past, then the host address is sent to `stdout`. The CPAN [CPAN] `Net::DNS` [NETDNS] module was the protocol engine for this utility and was perfectly suitable.

A BSD UNIX `crontab` command is used to run this utility on a nightly basis:

```
4 2 * * * /var/ra/expire.pl | \
    xargs /var/ra/delhost.sh
```

3.2.4. Monitoring Mail Server Logs – watchmaillog.pl

Our mail server ([POSTFIX]) is configured with the following `body_checks` rule:

```
/^UEsDBAoA/ DISCARD mailworm1
```

This rule is one of many that cause inbound mail to be checked for "worm" content. As indicated by the rule, such messages are discarded without notice to the sender. As a side effect, a message is sent to the `syslog` facility indicating that a message was discarded due to this rule, and telling the IP source address of the sending host. The `watchmaillog.pl` PERL utility, which is not reproduced here, reads the system log files using the BSD UNIX `tail -F` command and searches for markers indicating that "worm" content was discarded. These are parsed to find the IP source address of the sender, and the `addhost.sh` command

is run with the following arguments:

$1	host address
$2	mailserver hostname
$3	"watchmaillog"
$4	mailworm identifier

Assuming that host 220.192.60.187 sent a copy of mailworm2 to mail server sa.vix.com on 2004-02-28 at 23:44:57 (UTC), the resulting zone content would be as follows:

```
$ORIGIN 60.192.220.ra.vix.com.
187 A   0.0.0.0
    TXT "reason" "sa.vix.com" \
        "watchmaillog" "mailworm2"
    TXT "created" "20040228234457"
```

The watchmaillog.pl utility is started at boot time, just after the syslog and Postfix services.

Note that the Postfix body_checks rules are applied to every line of every inbound message. It's necessary for a mail server to have a very fast CPU, and very high main memory bandwidth, in order to apply a moderate number of rules to millions of messages per day. This is true of all virus and worm detectors, whether commercial, open source, or home grown.

3.2.5. Other Sources of Data

Other services such as the Apache web server also produce log files, and it is reasonable to assume that each such service ought to have its own log file watcher/postprocessor that uses addhost.sh to record worm sources.

We also have a customized intrusion detection system (IDS) that listens to a large subset of the IPv4 address space. Because this address space is "dark" (meaning that no domain names lead to it), the only connections we receive are from malfeasants who are randomly scanning the IP address space searching for vulnerable servers. Our customized IDS pretends to be vulnerable, but the only actual effect of sending us a malicious payload is to have the source address and payload logged in a database, and to feed the addhost.sh utility with new data about malicious endsystem host addresses.

In principle and by design, many other data sources could be added.

3.2.6. ISC BIND9 Configuration

In [ISCBIND] (version 9.3.0), there were three configuration activities necessary to make this project possible.

First, a new zone had to be created by hand, starting with just an SOA RR (start of authority record) and some NS (name server) records.

Second, a new cryptorandom key had to be created, using the following command:

```
dnskeygen -H 128 -n update. -h
```

Third and finally, the named.conf file had to be amended to include knowledge of this key and of this zone. That configuration file change is roughly as follows:

```
acl mynets {
  2001:4f8::/32;
  204.152.184/21;
  192.5.4/23;
};

key update-ra {
  algorithm hmac-md5;
  secret "84qHhYEAB7cP00OVt8YD6Q==";
};

zone "ra.vix.com" {
  type master;
  file "pri/ra.vix.com";
  allow-transfer { mynets; localhost; };
  allow-query { mynets; localhost; };
  allow-update { key update-ra; };
```

```
};
```

Let's walk through these configuration elements one at a time. `mynets` is an access control list covering all local IPv6 and IPv4 networks under the same local administrative control, and therefore trusted to be allowed to see the content of the `ra.vix.com` zone. It is important to restrict this data to trusted endsystems since accidental use could result in unwanted liability.

`update-ra` is the name of the key we generated using `dnskeygen`, and its `secret` is the KEY RR (key record) from the `/var/ra/Kupdate-ra.+157+43810.key` file, which must be securely copied and stored on every host system where the `addhost.sh` and `delhost.sh` utilities will be used. Since possession of this "shared secret" grants permission to update the zone, this file should be made readable only by the `root` or other privileged user id.

3.2.7. Postfix Configuration

In addition to the `body_checks` rules described in Section 3.2.4 above, it is necessary to configure every Postfix server in the local enterprise as a subscriber to your private RTBL. This is done in the `/etc/postfix/main.cf` file, in the `smtpd_recipient_restrictions` setting, usually after the standard invocation of `permit_mynetworks`, and ideally before any other RTBL subscriptions. Here is an example:

```
smtpd_recipient_restrictions =
  permit_sasl_authenticated,
  ...
  permit_mynetworks,
  ...
  check_recipient_access \
    regexp:/etc/postfix/exempt.regex,
  reject_rbl_client ra.vix.com,
  reject_rbl_client \
    rbl-plus.mail-abuse.org,
  reject_rbl_client \
    nonconfirm.mail-abuse.org,
  ...
```

Here we see a reference to an `exempt.regex` file, which contains a list of regular expressions to match those recipients who want to receive all of their e-mail, even if it's spam. It's because of this requirement that our RTBL subscriptions are declared in `smtpd_recipient_restrictions` rather than in `smtpd_helo_restrictions`. An example `exempt.regex` file would be:

```
/postmaster@/           OK
/abuse@/                OK
```

3.3. Difficulties Encountered

A number of weaknesses were exposed in both the protocols and their software implementations during the private RTBL project described in this section.

3.3.1. Negative Caching

Because of negative caching, a "worm train" of many inbound messages from the same source will not be immediately stopped by this technology. This is because an initial RTBL check will result in a cached "no such name exists" condition in the local caching recursive name servers. Any queries about this same domain name for the next few minutes will be answered from this cache, even if a dynamic update has caused new data to appear in the zone itself.

Negative caching is a necessary part of the DNS protocol, but it may be necessary to add another type of RTBL subscription directive to Postfix, which would attempt a dynamic update with a prerequisite of the name's nonexistence and no actual update operations. This would defeat negative caching by making a round trip to the authority server for all domain names in a specified RTBL. Such a feature could be dan-

gerously destructive toward DNS caching, and would need to be disabled by default.

3.3.2. Memory Limits

ISC BIND9 behaves poorly when it runs low on memory. All zone data in [ISCBIND] (version 9.3.0) is stored in the heap, including the currently served version of a zone, any prior versions still being accessed by an outbound zone transfer, and unfortunately, deleted records whose heap memory has not yet reused by new data. If the expiration process described in Section 3.2.3 is not run often enough, or if the cutoff date is so far in the past that not enough data is purged on a nightly basis, then BIND9 will enter a state where it cannot get enough heap space to do zone maintainance.

Recent experience has shown that a process memory limit of 1.5 Gigabytes (1,500 Megabytes) is enough to hold a private RTBL of about five million host addresses including the associated `reason` and `created` text records, assuming a 45-day expiry and a "class B" network (with 65,535 possible host addresses) feeding an intrusion detection device. Since longer expiry periods and more IDS contributors would yield a more accurate and more useful RTBL, these limits will have to be relaxed somehow.

Future work on ISC BIND9 will include non-heap zone data storage methods, as well as a more robust recovery process when heap memory is exhausted.

3.3.3. Zone Rewrite Granularity

[ISCBIND] (version 9.3.0) uses a journalling storage system to record zone changes. In the event of an untimely process death (such as a program or server crash) these changes can be reapplied to the zone at the next name server restart. If this journal grows beyond a calculated limit, then it is applied in real time by writing an updated zone to the file system and then truncating the journal. This design is technically correct, but creates two serious operational problems.

First, there is a lack of scheduler granularity during the zone rewrite process, such that some update requests can be lost due to server timeouts or even signalled server failures. Lost updates can include new worm data, or deletions due to the nightly expiration process. BIND9 will have to be improved to ensure that all updates are properly handled even during times of heavy background maintainance. Signalling a server failure is better than letting the client experience a timeout, just as an example of what "properly handled" means in this context.

Second, there is a hard upper limit on the number of updates BIND9 can accept per second, and this limit is enforced by storage hardware. The DNS UPDATE proposal [RFC2136] requires that all updates be committed to stable storage before an server responds to the update request. In BSD UNIX, this means an `fsync` system call will be made, which results in physical I/O. Even the fastest RAID5 storage systems on the market today can only do a few dozen or a few hundred physical I/O operations per second. Clearly, some way will have to be found to "batch" these I/O operations, which may mean bending or amending the DNS UPDATE protocol.

3.4. Future Work

In addition to the improvements mentioned in Section 3.3 above, work must continue toward the goal of packaging up these tools and publishing them for general use by the commu-

nity. Now that endsystem hosts who will be spamming you in the weeks to come are preannouncing their intent by trying to send identifiable and malicious payloads, only great good could come from a generally available method of shunning traffic from these hosts. This could make the attacks less successful overall, but could also assist with product liability lawsuits against monopoly providers of unsecure operating system platforms, or against malfeasants who take advantage of these insecure endsystems.

Some consideration is also being given to creating a robust, high availability, public RTBL based on these tools. With dozens or hundreds of trusted parties feeding the system and subscribing to it, it may become possible to so quickly and so thoroughly "shun" hosts running malicious software ("malware") and hosts who mindlessly forward these payloads, to provide a global economic disincentive to either own, operate, abuse, or provide the hosts responsible for almost all known forms and instances of Internet abuse as of this writing.

4. Conclusion

The Domain Name System has made a successful transition from a mostly static system whose content could only change due to external human action, to a vibrantly dynamic system whose content can change frequently and robotically.

This new dynamicism offers the possibility for new services to be delivered using DNS as a conduit. The data model offered to these new services includes a global query population, and moderately large update populations, with high coherency, reliability and performance.

At least one new application has been built using the DNS Services model, and the results so far are very encouraging. The DNS Services model is effective, and we hope that this paper will encourage more applications to use this model.

References

[CPAN]
Comprehensive PERL Archive Network. URL *http://www.cpan.org/*.

[ISCBIND]
ISC BIND Home Page, Internet Systems Consortium, Inc. (ISC). URL *http://www.isc.org/sw/bind/*.

[MAPSRBL]
P. Vixie. MAPS RBL Usage, Mail Abuse Prevention System (MAPS). URL *http://mail-abuse.org/rbl/usage.html.*

[NETDNS]
Perl interface to the DNS resolver. URL *http://www.net-dns.org/*.

[POSTFIX]
Wietse Venema. The Postfix Home Page. URL *http://www.postfix.org/*.

[RFC1034]
P. Mockapetris. Domain Name System – Concepts and Facilities, IETF, 1987.

[RFC1995]
M. Ohta. Incremental Zone Transfer in DNS, IETF, 1996.

[RFC1996]
P. Vixie. A Mechanism for Prompt Notification of Zone Changes, IETF, 1996.

[RFC2136]
P. Vixie, et al. Dynamic Updates in the Domain Name System, IETF, 1997.

[RFC2845]
P. Vixie, et al. Secret Key Transaction Authentication for DNS, IETF, 2000.

[RFC2931]
D. Eastlake. DNS Request and Transaction Signatures (SIG(0)s), IETF, 2000.

[RFC3645]
S. Kwan, et al. Generic Security Service Algorithm for Secret Key Transaction Authentication for DNS (GSS-TSIG), IETF, 2003.