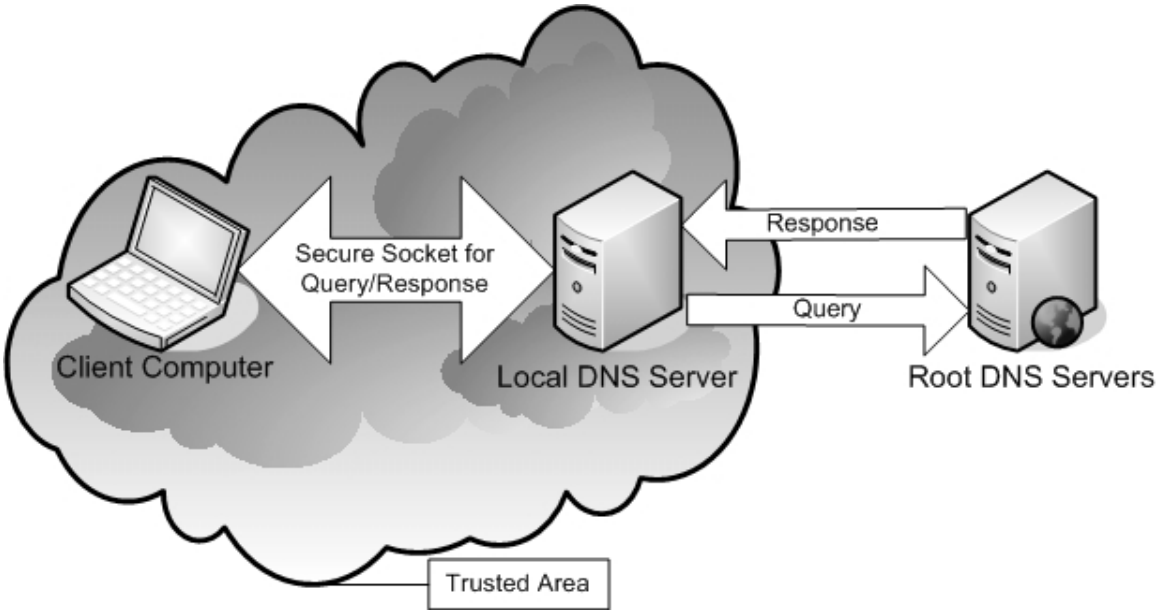
RASD – Rapid Adaptive Secure DNS

Matthew Weaver
Jeremy Witmer



Dr. Chow, Advising
CS 622 – Distributed Networks
University of Colorado, Colorado Springs
Fall 2007

Abstract

The rapid growth of the Internet and internal networks has greatly expanded the amount of private financial, medical and other confidential data moving electronically. A rise in phishing, botnets, DDOS attacks, and other assaults aimed at obtaining this data has shown that DNS needs a level of security that it does not currently have. In this project, we have designed and implemented a scalable system to secure DNS traffic on local and larger networks, and provide a means to rapidly update cached client DNS records.

The local RASD client is implemented to keep all unsecured UDP traffic on the client, and uses XML over SSL sockets to exchange data between the client and the server. In testing, the initial DNS requests on the RASD system take approximately 2.8 times as long as the normal Windows XP DNS client. However, because of the implementation of the local caching for name records in the RASD client, it outperforms the Windows client in repeated requests for the same hostname by up to 50%.

Table of Contents

Table of Figures

Introduction

Internet usage for e-commerce and business continues to grow at a rapid pace. Unfortunately, this significant increase in personal, corporate, and financial data moving over the internet has brought with it a corresponding increase in activity by third parties aimed at obtaining that data for their own profit. The rise of DDOS attacks, phishing, and other activities has made the flaws with the Domain Name System apparent. As DNS is implemented today, holes allowing for DNS hijacking and other problems stem from the fact that DNS records are not cryptographically signed, and are not transmitted over trusted channels. Furthermore, each client on the system carries its own value for expiration on a name record, the Time To Live (TTL), and with the many varied implementations of TTL systems, client programs can hold on to and continue to use expired name records. This limitation also makes changing a name record a somewhat haphazard and patchwork process. See [1] for further information.

To address a subset of these concerns, we propose to design and implement an extension to existing DNS services to allow for more secure, and more adaptive DNS behavior. This extension is envisioned as an implementation of a secure DNS architecture to allow for secure exchange of trusted domain records, setting up encrypted channels for DNS information exchange, and to allow rapid updating of DNS records to rapidly adapt to shifting network conditions/outages and DDOS attacks. This extension to the existing DNS system will allow client computers to rapidly route around outages, and more direct network attacks.

This project has two goals. First, we seek to build a trust relationship from the DNS servers to the local DNS client on each machine, allowing the user-space programs to implicitly trust the DNS information they receive (See [2] for further overview on this.) This is done by setting up a secure channel between the DNS client on the local machine and the DNS name server.

The second goal of the project is to allow for rapid adaptability from the client perspective. In the event of a DDOS attack or other major network outage, the DNS system cannot currently adapt rapidly to change domain name to IP address mappings across all their clients to re-route traffic. RASD will allow the DNS servers to push new name records to clients. Clients can also be set to regularly check for new name records, either globally, or for specific name records. This will operate much like TTL currently does, but also allowing the server to push new records will eliminate problems with long expiration times and hanging records. This will allow for rapid updates to name records in all conditions, and can allow for implementation of the SCOLD protocol, as proposed in [3].

RASD replaces the DNS resolver on each client computer with a new client. All DNS requests from the local machine will then go to the RASD client. The RASD client has a local list of trusted DNS servers, and will only query for name records from those servers. The RASD server now performs two functions. First, it provides normal DNS records for requests, as normal DNS servers do today. Second, the RASD server can push

updates to the clients, both to update DNS records (effectively overriding any kind of TTL limitations), and to add new RASD servers to the client's trust list. To allow for failover capability, each client should have more than one RASD server in its trust list, and each client can ask for new RASD servers from its current list of trusted servers, allowing a constantly expanding tree of trusted servers to be created. Each RASD server can also send/receive updates to its own parent servers, up to a set of root servers.

In pursuit of these goals, the RASD project seeks to add adaptability and security to the existing DNS system, without significantly increasing the load on each DNS server. RASD servers and clients will require more traffic than existing DNS, but this project seeks to extend DNS without raising the amount of bandwidth required above acceptable levels.

RASD System Design

The RASD system is designed in two parts, a DNS request handler that sits on the client computer (referred to collectively as the client), and the replacement DNS server, to which the client connects.

Trusted Name Record Information

The most significant problem with the current design of the DNS system now is that the name client's ability to trust any given name record basically stops at their computer. Figure 1 shows a basic view of the DNS system as it currently stands.

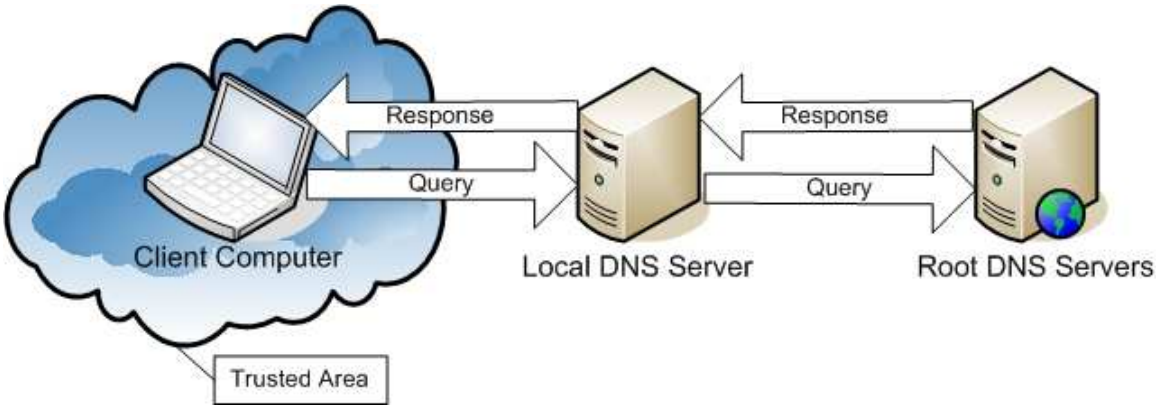


Figure 1: Basic DNS Architecture

The client computer transmits its DNS query for a domain name to the local server, which replies, or passed the query on to the next level of servers, as far as the actual DNS root servers, which serve as the ultimate authority for DNS on the Internet. The query systematically moves up the chain through the DNS servers until it reaches a DNS server that has the necessary domain name record, matching the domain name to its public IP address. Starting at that server, a response is transmitted back down the tree to the client computer.

The largest issue with this system is the assumption of trust. DNS clients are written to assume that the domain name record (name record) received from the server is

trustworthy. However, since there isn't a trusted channel between the client and the server at any level, third parties can subvert the name record somewhere in transit, and insert a different IP for a domain name, or otherwise poison the name record, allowing them to deceive the client computer for a variety of reasons.

The RASD system seeks to add a level of trust to the system. Figure 2 shows the RASD extensions to the basic DNS architecture from Figure 1.

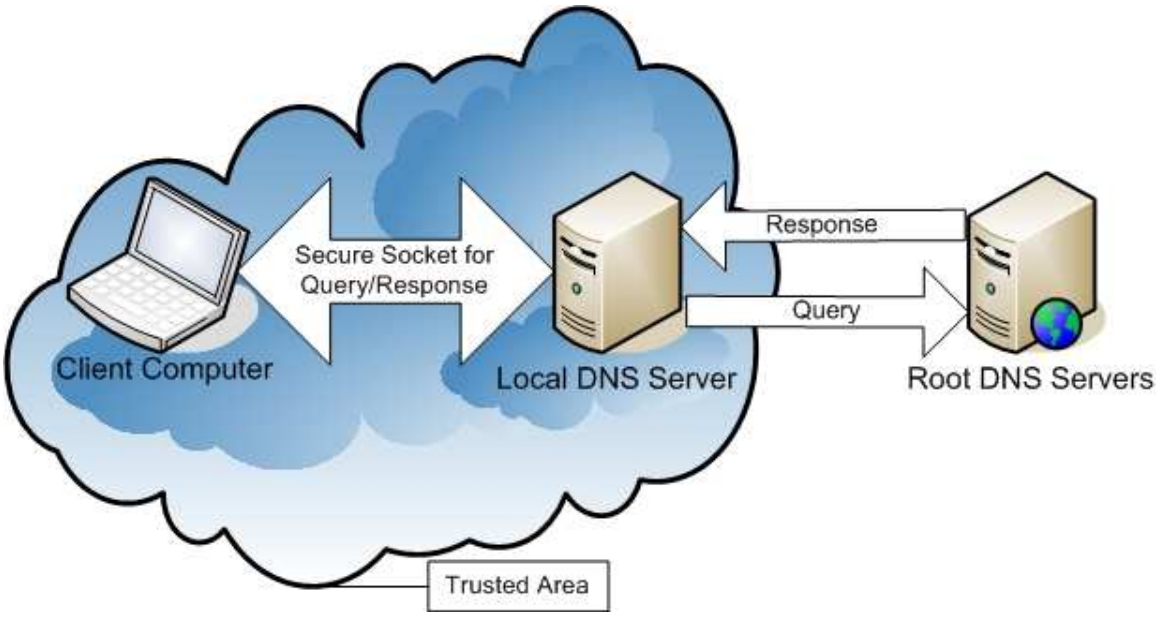


Figure 2: RASD DNS Architecture

RASD seeks to extend the trusted area for name record responses. Instead of using normal UDP connections from the client to the local DNS servers, RASD uses SSL for any connections outside the client computer. Any unsecured UDP traffic is contained to the client computer. Since the name record responses (and even the DNS queries) are encrypted, the client can trust the data returned from the server. As designed currently, the trusted channels have to end at some point within the tree of DNS servers and data will be passed unencrypted, but this issue is too extensive to address within the confines of this project. RASD is envisioned to work at the corporate/university/small ISP sized-network level, to protect individual client computers from DNS hijacking/poisoning attacks. We assume that the main name servers within a network of this size have better security than the individual client computers, and are more resistant to attack. That being said, however, this system could potentially be extended to provide trusted channels all the way to the root DNS servers.

Rapid Updates of Name Records

The second problem the RASD system seeks to solve is the current lack of rapid adaptability of the current DNS system to rapidly changing network conditions. DNS clients are normally implemented to cache name records locally to speed up query time. Each name record result returned from a DNS server has a Time To Live, or TTL value,

that dictates when the name record should expire from the client's cache. Unfortunately, various DNS clients handle the TTL value in numerous different ways, resulting in systems that can keep old name records in their cache long after they should be expired.

Furthermore, as the internet becomes more and more complex, and more and more significant transactions and important data flow over the internet, the need to be able to rapidly adapt to changing network conditions becomes a priority. No matter what actually causes the outages, whether intentional DDOS attacks, or network failures due to natural causes, we propose a system that will allow systems to rapidly route around non-responsive sections of the network by updating name records.

In the current DNS architecture in use today, name records are only updated when a client requests the name record from a server. In our system, each server has a list of known clients, and can push updates to those clients. The clients then ignore any TTL values and use the updated name record immediately.

For example, in an attack scenario proposed by the SCOLD paper in [3], the main routing gateway to an application is under attack. In the SCOLD scenario, backup routers come online and non-attack traffic is routed to the application server. When the attack starts, the intrusion detection system (IDS) sends an alert message to the Reroute Coordinator, indicating that the main route to the application servers is under attack, and traffic needs to be routed to the backup systems. We propose the RASD system would be ideal for re-directing clients. The updated name records can be pushed to the clients, and the clients will immediately start using the new name records for their requests. Also, even though the network is under attack, the clients can implicitly trust the new name records, because the DNS servers are using a secure channel.

Data Exchange Format

This section of the document covers the formats used to exchange data on the client and between the client and the server.

DNS Query/Response Packet

DNS request/response packets are UDP packets normally sent to port 53. The client requests are sourced from a random port above 1024, with a sequence number/id used to uniquely identify the responses. When the server responds to a query, the destination port is the same as the random source port from which the query is initially transmitted. For more extensive details, including the actual format of the packets, see the list of RFCs pertaining to DNS at http://www.dns.net/dnsrd/rfc/.

RASD Client/Server Interchange

All data exchange done over the secure socket between the client and server is done in XML, in a schema designed for this project. For simply queries and updates of A records, the client and server interact using the following XML formats. First, to request the DNS record for a domain name, the client sends the server the following XML:

```
<interchange>
```

```
    <queries>
        <query alias="www.wikipedia.org” type="host address"
class="INET"/>
    </queries>
</interchange>
```

The client here has requested the domain name record for www.wikipedia.org, and the type and class fields are also included, so that the prototype system can be later extended to handle more than just the A record. When the client queries for an updated name record, the server responds with the following XML packet.

```
<interchange>
    <updates>
        <update alias="www.wikipedia.org" ip="66.230.200.100" ttl="60"
order="0"/>
    </updates>
    <queries>
        <query alias="www.wikipedia.org” type="host address"
class="INET"/>
    </queries>
</interchange>
```

As with a normal DNS server, the RASD server here returns a packet that contains both the original query information, and the name record information for the requested domain. The <updates> section of the response will contain as IP addresses as are available for the domain in question.

Finally, the server can push an <interchange> packet to force the clients to update their cached name records independent of any TTL restraints. The server sends the following XML packet:

```
<interchange>
    <updates>
        <update alias="www.wikipedia.org" ip="66.230.200.100" ttl="60"
order="0"/>
    </updates>
</interchange>
```

Again, as with the response packet, the server can package numerous updates into a single transmission. The client will parse this packet and immediately start using these TTL values.

Due to time constraints, the prototype RASD system currently only supports the exchange of normal A records, but because the exchange format is simple XML, we can easily extend it to handle MX records, IPv6 AAAA records, etc. Both the <query> and <update> formats are very flexible, and can be extended as necessary to accommodate all forms of DNS traffic.

Again, due to time constraints the <interchange> packet at present does not currently return the authoritative name server, or name server information at all, but the <update> packet can easily be extended to contain this information.

Software

This section of the document discusses the design of the client and server software, along with details of the actual prototype implementation of the RASD system. Both the prototype client DNS request handler (the "client") and the prototype RASD DNS server (the "server") are written in Java.

Client Software

The following flow-chart outlines the behavior of the RASD client.

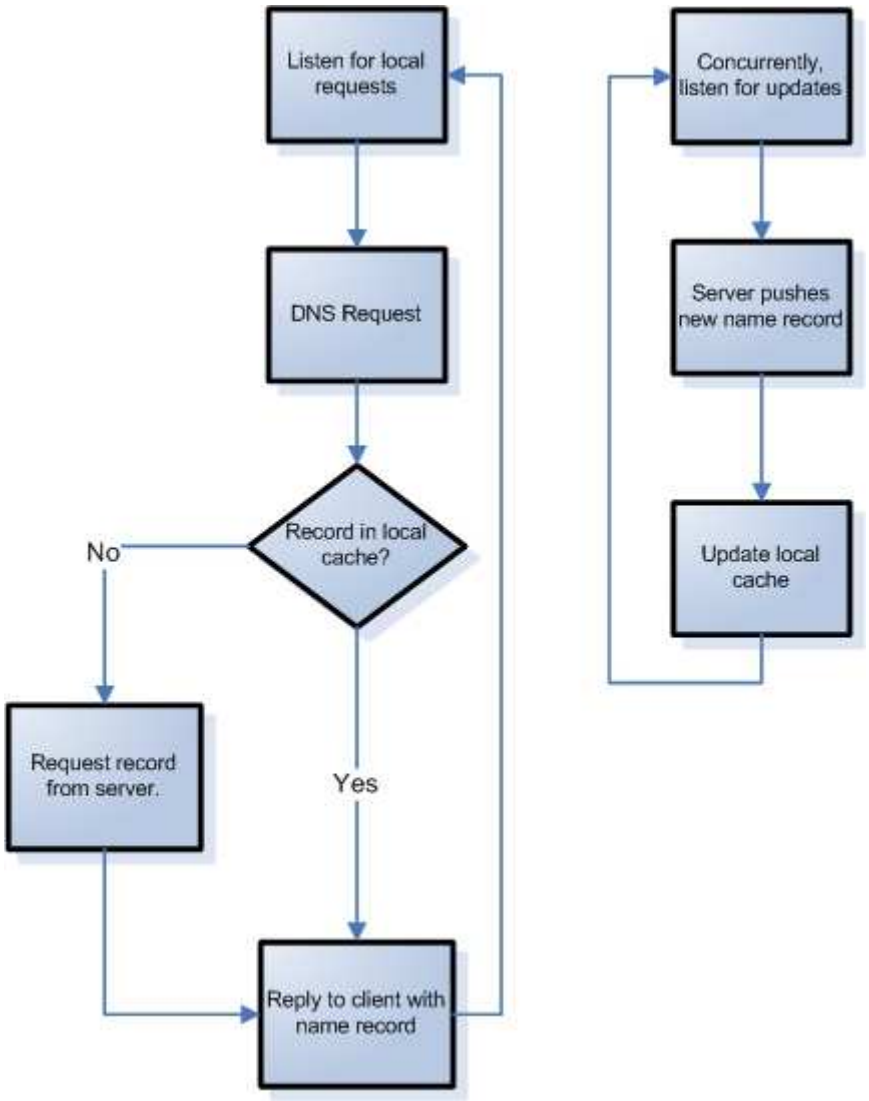


Figure 3: RASD Client Operation Flow

The client runs a pair of processes concurrently. The first process listens and responds to DNS queries from other processes on the same computer. The second process listens for pushed updates from the server.

All of the UDP packet handling is done by the client, including the storing of the sequence numbers and source ports for DNS requests, so that responses from the server can be sent to the correct requestor.

The client stores a list of addresses for one or more RASD DNS servers. While the client is constantly listening for queries from the local software, and for updates from the server, the client does not maintain a constant connection with the server. This is to prevent the server from having a significant number of open connections, because the client/server traffic is done over UDP.

When a DNS query is received by the client, it parses out the datagram packet and stores the request, the sequence number, and the source port for the requesting process. The client then checks its local cache for a matching name record. If one exists and hasn't expired, the client immediately responds to the requestor. If the record is expired, or the client does not have the necessary record in the local cache, the client makes a connection over SSL to the RASD server and requests the name record. When the server responds, the client assembles the reply datagram packet, and sends it to the requesting process.

In this architecture, the client chooses the server to which it wishes to connect, leaving the decision about which server to trust on the client side, assuming that the server has the necessary keys to connect to the server over SSL.

In practice, the client computer runs the RASD client process, and sets its own DNS server to 127.0.0.1, localhost. All UDP DNS queries are routed to the local RASD client on port 53.

Server Software

The RASD server behaves according to the flow chart in figure 4. As with the client, the RASD server runs two different operations concurrently. The first operation listens for client DNS queries and responds, and the second process waits for updates to its local name records, and pushes those updates to the clients.

Initially, a server starts out without a list of clients. As discussed in the client section, the clients choose the RASD server, and make the initial SSL connection, provided that the client has the correct key to authenticate over SSL. Once a client has connected to a server, that server then stores the client information, including port and IP address, in its internal list of clients. Whenever the server gets an updated name record, it then pushes that information to all the clients in its list.

When a DNS query <interchange> packet is received from one of the clients, the server process the request, and then looks up the requested host name in its own cache. If the server has the requested name record in the cache, it generates a reply <interchange>

XML packet, and returns it to the client. If the name record is not in the server cache, the server requests the information from the next level server. This is where the variability designed into the RASD architecture comes into play. The next level server can either be another RASD server, in which case the local server requests the name record over an SSL connection. If the next level server is a normal DNS server, the server can translate the request back to UDP and transmit is as a normal DNS request. Under either circumstance, the server will receive back a name record, package it into the <interchange> packet, and return it to the originally requesting client.

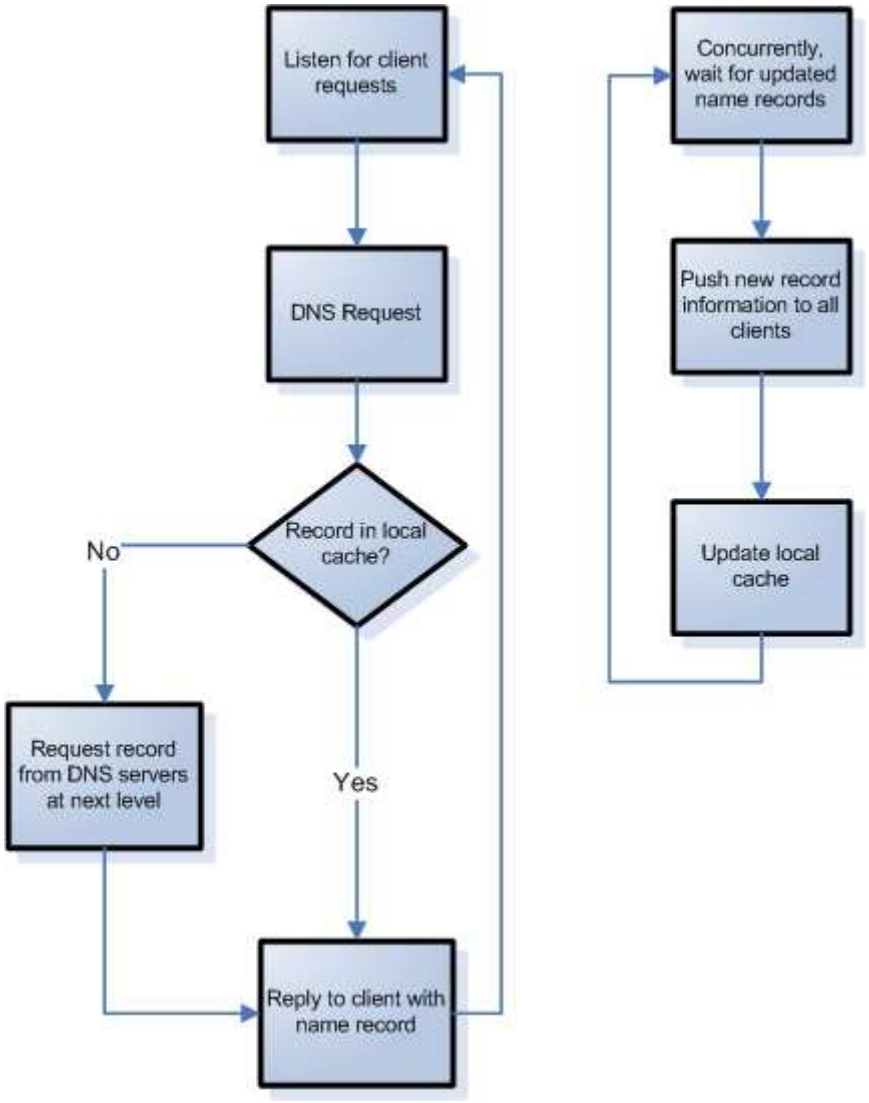


Figure 4: RASD Server Operation Flow

One of the more important architecture decisions between the client and the server is the shifting of the DNS traffic from UDP to TCP. While the DNS specification does allow traffic over TCP, normal DNS traffic is over UDP. However, to enable the use of SSL security on the information, the traffic here is shifted to TCP. Since UDP traffic requires less bandwidth than identical TCP traffic, this extra bandwidth overhead may become an

issue as the number of RASD clients using a single server becomes large. This has not been tested to date.

Prototype Test Results

Two types of tests were run comparing a client using the RASD server to a client using a normal DNS server, in this case, the Colorado Springs Qwest primary DNS server.

The first test used the DNSTester software from [4] to directly compare the performance of 30 random DNS lookups between queries going through the RASD client to the RASD server and the normal Windows DNS client. Table 1 presents the results.

Hostname	RASD Lookup Time (s)	Windows client lookup time (s)
angelfire.com	0.468	0.25
tripod.com	0.421	0.25
geocities.com	0.609	0.062
topix.com	0.562	0.109
youtube.com	0.531	0.078
amazon.com	0.828	0.109
webersports.com	0.781	0.328
myspace.com	0.75	0.0078
flyordie.com	0.703	0.171
snowshack.com	0.656	0.234
bizrate.com	0.734	0.078
dpmhi.com	0.64	0.453
mhicorp.com	0.484	0.234
plantationhomes.com	0.687	0.234
mhihotels.com	0.593	0.171
go.com	0.078	0.109
uschamber.com	0.296	0.156
ncfcorporation.com	0.515	0.234
homestead.com	0.343	0.156
flickr.com	0.25	0.109
ncf.com	0.468	0.234
stockmarketenews.com	0.546	0.234
petroflexna.com	0.593	0.234
pnanet.com	0.5	0.234
nia.com	0.546	0.25
agilent.com	0.406	0.062
peyamner.com	0.359	0.062
yahoo.com	0.156	0.078
flbb.com	0.859	0.468
blogspot.com	0.671	0.234
AVERAGE	0.534	0.187

Table 1: Random Lookup Results

This table shows that, on average, queries through the RASD system take approximately 2.8 times as long as queries through the normal windows DNS client for an un-cached

lookup. The overhead here is from translating the data to XML, and setting up the SSL sockets for the communication.

However, the RASD client also does local caching, which increases the performance over time. The second test with the prototype compared five hostnames over a series of lookups to the same hostname in a small period of time. Queries were sent for each hostname 10 times within a two minute period using both the normal windows client and the RASD system. The Dig software from [5], which shows all the information about a DNS query, including the total query time, was used to collect the numbers. Table 2 shows the 10 query times for just google.com, and table 3 shows the average results for all five domains.

google.com Lookup	RASD Lookup Time (s)	WinClient Lookup Time (s)
1	0.153	0.125
2	0.031	0.062
3	0.031	0.062
4	0.031	0.046
5	0.031	0.046
6	0.015	0.062
7	0.031	0.046
8	0.015	0.062
9	0.015	0.062
10	0.015	0.093
AVERAGE	0.0368	0.0666

Table 2: 10 DNS Queries for google.com

Domain Name	RASD Average (s)	WinClient Average (s)
google.com	0.0368	0.0666
compusa.com	0.0342	0.0728
agilent.com	0.01475	0.0635
amazon.com	0.0244	0.0604
yahoo.com	0.0229	0.0524

Table 3: Average Time for 10 Queries

As can be seen from table 2, the initial setup takes longer with the RASD client, but once the name record is in the local cache, all future lookups are faster. The results in table 3 show the resulting 50% or better performance from the RASD client across a number of domains. Over time, the RASD client outperforms the Windows client.

Testing Issues

We were only able to test the RASD client against the standard Windows client, and only on a small scale. For more extensive data, the RASD system should be compared to a range of DNS clients, including the standard Linux client, and DNSSEC enabled clients. We also need to test the server under the load of a significant number of clients, as it would be used in a normal network. However, despite the limited testing, we believe that the architecture has proved itself.

After analyzing the code, it appears that most of the overhead in the client is with the setup and teardown of the secure sockets. The current prototype opens and closes a number of sockets during the exchange of data, since both the server and client are multi-threaded. With further work and design, the sockets should be able to synchronize and share among the threads, eliminating the need to open and close sockets. This would eliminate the setup/teardown time for each transmission.

Though the client cache means that repeated queries return quickly, the overhead in the XML exchange slows the client/server communication down. With further work and refactoring, this overhead could also be reduced.

Further Research

Extend DNS Message Types

This project served as a proof-of-concept using only INET Class, Type A records. To actually be considered for deployment, the RASD prototype needs to be extended to handle the full spectrum of DNS record types and classes. This is a fairly straightforward process, though somewhat involved. The message types and communications handlers in the RASD prototype were designed to be easily extensible, but DNS has enough message types to make testing to ensure coverage a fairly involved process.

This extension of the existing prototype must be the first consideration in further work, so that RASD can be fully tested for performance against existing DNS servers.

RASD Server Discovery

To make the RASD server a useful replacement for DNS, the client software should be extended to automatically discover servers. The client would be able to scan for RASD servers within the current subnet, and automatically connect, assuming that the client can authenticate with the server.

The server software would need to be modified to listen for discovery requests from clients, and to reply with a packet containing server information and configuration data.

Automatic RASD Client Installation/Configuration

Normally, networks configured for DHCP automatically assign and IP address to new client computers connecting to the network, and also provide DNS server information to the client. In a network utilizing RASD servers for DNS, the RASD server could provide the localhost address, 127.0.0.1, as the DNS server address, and also provide the client the ability to download and run the RASD client. If the client computer already has the RASD client installed, the DHCP server can simply direct it to use the RASD client for DNS queries. This process would allow clients to use the RASD system with little or no user configuration required.

SCOLD Environment Testing

As suggested earlier in this paper, the ability of the RASD server to do rapid pushing of updated name records may provide an implementation for SCOLD network defense. This would require setting up the network infrastructure and DDOS testing system along with a RASD environment.

Standardized DNS Record Caching

Currently, both the RASD server and client cache name records internally, in Java data structures. To more closely conform to the operation of normal DNS servers, the name record information should be stored in text files, for persistence across instances of the programs, and for easy access by other programs.

Conclusions

Though the prototype server and client written for this project do not handle the full range of DNS record types, the prototype shows that this architecture is valid for a secure DNS system. While the prototype needs extension, and the prototype code needs some refactoring to reduce the communications, the basic architecture and data exchange format are sound.

With further research and development, the RASD system is a viable option for secure DNS, with many options for future research.

References

[1] A. Friedlander, A. Mankin, WD Maughan, and S. Crocker. "DNSSEC: A Protocol Towards Securing the Internet Infrastructure". Communications of the ACM. Vol. 50, Num. 6. pp 44-50. June 2007.
[2] G. Ateniese and S. Mangard. "A New Approach to DNS Security (DNSSEC)". Proceedings of the 8th ACM conference on Computer and Communications Security. pp 86-95. 2001.
[3] C.E. Chow, Y. Cai, D. Wilkinson, and G. Godavari. "Secure Collective Defense System". Global Telecommunications Conference (GLOBECOM '04). Volume 4. pp 2245-2249. December 2004.
[4] Website: “DNS Tester”. http://www.codeproject.com/KB/IP/DNSTester.aspx.
[5] Website: “Dig DNS Query Tool“. http://members.shaw.ca/nicholas.fong/dig/.