Smart Computing Review

HTTP-based Smart Transportation of DNS Queries and Applications

Aziz Mohaisen[1] **and Manar Mohaisen**[2]

[1]CSE Department, SUNY Buffalo / NY, USA / mohaisen@buffalo.edu

[2]School of EEC Engineering, KoreaTech / 330-708, Cheonan, R. of Korea / manar.subhi@koreatech.ac.kr

* Corresponding Author: Aziz Mohaisen

Received May 17, 2015; Revised June 22, 2015; Accepted July 20, 2015; Published August 31, 2015

Abstract: In this paper we introduce a system, called DJSON, which enables HTTP transport of Domain Name System traffic. DJSON enables re-encoding of the existing Domain Name System message format, so that it can traverse hostile territory with confidence without modifying the underlying Domain Name System design. In DJSON, Domain Name System messages are sent and received with a properly formatted HTML using a JSON encoding that allows bidirectional mapping to and from traditional Domain Name System transport encodings. This guarantees that interoperability is no worse than it is today. HTTP can be used to work around the problem where middle boxes have interoperability problems. DJSON aims to solve several real-world and operational problems. DJSON is designed to "bridge" Domain Name System across areas where Domain Name System packets might be mangled, deliberately modified or blocked. DJSON further aims to enable and address improved reliability, availability, and security. Detailed discussions, experiments run on a prototype of DJSON, and analysis show the effectiveness and relevance of our work.

Keywords: DNS, Security, Privacy, Network, HTTP

Introduction

The Domain Name System (DNS) is a distributed database that has enabled a massive growth of the World Wide Web ("the Web"). DNS provides the mappings of human readable domain names (such as www.example.com) to Internet Protocol (IP) addresses. The Internet [11] is considerably older than the Web [10], but the Web has been the principal driver in its growth.

DOI: 10.6029/smartcr.2015.04.001

The authors would like to thank Brian Dickson for helping with the implementation of DJSON. This work was done mainly while the first author was with Verisign Labs. The views and opinions expressed in this paper are the views of the authors, and do not necessarily represent the policy or position VeriSign, Inc.

Malicious activity has revealed vulnerabilities not accounted for at the time of DNS's original design [15]. While DNS Security Extensions (DNSSEC) addresses many of these, end-to-end adoption is needed. New DNS features are not well supported at their inception. Upgrades and extensions are often adopted slowly by large enterprises due to non-DNS obstacles. Numerous ancillary systems do not support DNS adequately or at all. Another problem in the environment surrounding DNS is that DNS is trivial to divert through a rewriting middle box. Consumer ISPs and public WiFi frequently rewrite DNS [21]. This is becoming the norm, but it interferes with new features. In the hands of state operators, it is a major technique used for censorship. We contend that DJSON, through its use of HTTP, could change the playing field of DNS re-writing and take some control of DNS out of the hands of various types of intermediaries.

Contribution. Our contribution is as follows. First, we outline the case for HTTP transport of DNS traffic, which is motivated by several real-world deployment aspects and settings. Second, we introduce the design, implementation, and validation of a system that enables HTTP transport of DNS traffic. We analyze DJSON, and present several measurements on a fully-functioning prototype. Our measurements demonstrate that incremental overhead of DJSON to DNS resolution is negligible. We further show that DJSON is incrementally deployable in the real-world Internet.

Organization. In Section II, we go over the motivation behind DJSON by considering several limitations and shortcomings in the Internet today that DJSON could help solve. In Section III, we introduce the preliminary findings of our work, including an overview of DNS and JSON. In Section IV, we introduce related work. In Section V, we review the design of DJSON which is followed by implementation details presented in Section VI. In Section VII, we introduce measurements, which are followed by a discussion in Section VIII. Concluding remarks are drawn in Section IX.

Motivation and Applications

The main goal behind us undertaking this research is to find ways to enable end hosts to use DNS with high confidence in their DNS service. The transport of DNS should not trade low overhead for a high risk of tampering. It is critical for any such transport to be capable of passing through intermediate systems (aka *middle boxes* [20]). The need to upgrade hosts without upgrading middle boxes tends to rule out the introduction of a novel transport protocol [19]. These issues are detailed as follows.

Lack of vendor support: In some cases, the software is an embedded element of a piece of hardware, such as a consumer-grade firewall or router, and termed firmware. Development of important new features is possible only when all systems are upgraded. Vendors are typically under no obligation to provide new features. The options in those cases are not pleasant: forego the new functionality, or replace the systems. We ideally want our system to operate on a highly-supported protocol on all platforms to address this issue.

Deployment scalability issues: Large enterprises may have to weigh a variety of factors in deciding to upgrade systems, including physical access to machines and time. Upgrades also carry risks, including failed upgrades and extended outages, the risk of introducing incompatibilities or instabilities, and potential training issues. Ideally, we want to use a system that does not require end-hosts involvement for its upgrade.

DNS Rewriting by ISPs and Public WiFi Operators: Ordinary DNS is completely vulnerable to manipulation by Man in the Middle (MitM) payload manipulation. The rise of the search engine business model has unfortunately created an incentive for parties to conduct MitM manipulation. The main method involves intercepting DNS lookup responses, which indicate that a name does not exist [17], and to modify those to redirect clients to monetized services [21]. Ideally, we want to implement a system that makes it harder or even impossible to rewrite without detection.

DNS Rewriting or Censorship by State entities: In some cases, the MitM may be a state (nation state government or government-sponsored entity), with a reason more ideological in nature [6]. In these cases, the state may wish to censor parts of the DNS namespace. Similarly, states may wish to replace content on the web via DNS redirection. Similar to the previous case, we want to implement a system that makes it harder for the censor to find or modify DNS traffic.

Primer on DNS and Name Resolution

It is helpful to first briefly describe the components, design, and operation of existing DNS. The DNS consists of the following elements:

- Stub resolvers (end user host systems)
- Recursive resolvers (caching name servers), sometimes called iterative resolvers
- Authority servers (including root and TLD servers)

An application on an end user's computer, such as a web browser, typically takes user input (from keyboard, or via URLs), and looks up Domain Names in order to access information through an IP address. The resolution process is usually carried out by the host's stub DNS resolver. The stub resolver converts the requested information into a DNS wire-format packet, sends it to the recursive resolver, and waits for the reply. A recursive resolver performs the bulk of the work: it maintains a cache of previously retrieved DNS values, and it performs the traversal of the DNS tree.

If the recursive resolver gets an answer, it sends it to the stub, which extracts the relevant data and returns it via the API. If it receives a DNS message indicating the name does not exist, it returns this to the stub. If at any point the recursive resolver fails to get any answer from a particular DNS server, it exercises its robust features — it may try the same server again, or look for another server. If it is unable to complete its task, it will report its failure to the stub.

Related Work

This paper stands on the shoulders of several other pieces of work related to DNS over HTTP. Aside from DNSSEC-trigger, the uses are for the display of DNS in web settings, rather than for transport of DNS from querier to servers and back. Below is a brief summary of past efforts:

- Multi-location DNS Looking Glass - Tool for performing DNS queries via RESTful interface in multiple locations, returning results in JSON format.
- DNS Looking Glass - Tool for performing DNS queries via RESTful interface, returning results in JSON format.
- DNS JSON - Source code project from circa 2009, partially developed but incomplete/abandoned. More details on the difference between this work and DJSON are listed below.
- DNSSEC-trigger - embedded control function in NLNetlabs' Unbound resolver, for attempting DNS queries over TCP port 80 when DNSSEC problems are encountered.
- Several other web-based DNS lookup tools.

Comparison: We **compare** those systems and initiatives with our work. There has been at least one previous effort to develop code for a DNS-JSON encoding [2], which appears to have been abandoned after one-way encoding was done in 2009. Another DNS JSON tool [3], similarly focuses only on answers, with a limited number of type codes. Yet another tool for looking up DNS via HTTP with JSON response is "dnsrest" [5]. It also focuses only on answer values. The "DNS Looking Glass" [1] is primarily designed for returning DNS answer data. Its JSON scheme is organized around DNS resolution metadata. The "Multilocation DNS Looking Glass" [4] uses a RESTful query mechanism of "node/qname/qtypename". The JSON response format is generic, encapsulating all types as string data. None of the schemes above facilitate decoding, and thus are not suitable for bidirectional use. DNSSEC-trigger [18] is a feature of the Unbound DNS implementation. The creator, NLNET Labs, has developed a group of heuristics to determine if there is an environmental obstacle to DNSSEC and offers a server that will parse and forward DNSSEC queries and responses over HTTP. It does not deterministically create a gateway in the manner that DJSON does. It is single-vendor, requires that the host itself runs Unbound, and requires TCP access to another Unbound server.

Other DNS presentation tools: A variety of web lookup tools exist, predominantly producing DNS validation (zone structure and hierarchy), maps, metadata, or literal output from the "dig" tool, in formats as varied as the purpose of the tools. Dig output, while being reasonably deterministic, is not sufficiently well-formed so as to facilitate *screen scraping* as a parsing method.

Our system: In contrast, the JSON scheme presented here, and used in DJSON, is designed for bidirectional mapping with well-defined mappings on all elements, the preservation of the ordering of elements, and fidelity-preserving metadata included in the JSON representation. This metadata provides a completely deterministic decoding capability, along with a *server* mode that has the capability to perform efficient one-pass mapping. Additionally, the JSON encoding maintains full readability by using the traditional *presentation* format of DNS fields, with each field labeled and structural relationships represented by the *array* encoding type.

Design

The system consists of two main components: a client and a server. Operation of the system requires that each client be configured to connect to a specific server. Servers are designed to support any number of clients (limited by the laws of physics), and there are no restrictions on the number of servers or their geographic placement. The client translates the requests it receives from the host on its DNS subsystem into JSON format, and forwards the query (over HTTP[s]) to the server. The server then receives the request, translates the request into DNS, and sends the DNS request to the server's configured DNS resolver. The DNS resolver answers the query and sends back the DNS format to the server, which translates the DNS answer into JSON format and sends it back to the DNS client stub. The client then returns the content received from the server.

Functional Design: The design of this transport mechanism necessarily needs to dovetail into the overall design and operation of the existing DNS. In particular, one critical requirement is that whatever it does needs to be transparent beyond its own borders. DNS requires complete interoperability, and this places strict rules on both the design and the implementation of any new DNS-related system. In order to provide for the highest degree of fidelity and interoperability, the decision was made to operate in a *bridge* (or *tunnel*) mode. Any place a DNS stub could communicate with a resolver is a place that a pair of gateways can be interposed. This high-level design allows a great degree of flexibility and scalability in deployment scenarios.

In the following, apart from the components of the system, we highlight some of our design options and explain the rationale for choosing JSON over other alternatives.

■ Queries: REST vs POST

A design choice that needs to be explained is the method by which the query is converted to an HTTP request sent by the client. There are two basic options: REST, which needs the query to be composed into the URL of the request, or POST method, which enables the use of JSON for representation. JSON + POST permits arbitrary data to easily be encoded.

In the overall design, there was a need to both encode and decode DNS messages to achieve the full gateway function in the direction of DNS Answers. Note that the DNS message format is identical for queries and replies. Given that JSON has one of the largest support and use bases on the Web, we selected it as our representation method. So, by using JSON + POST encoding for queries, all potential development cycles needed for designing, implementing, and debugging the REST functionality were avoided. Since the encode/decode process maintains DNS message fidelity, all current and future DNS query methods and options are automatically supported.

■ Client Side: Topological Possibilities

Table 1 shows a non-exhaustive list of ways in which a DNS-over-HTTP subsystem could be incorporated into the DNS resolution process. Each of those design options would enable certain level of design and maintenance complexities, and would answer the limitation posted in Section II on different ways. For example, implementing the client at the browser or client side, while still hiding DNS traffic in the HTTP volumetric traffic, is costly for maintenance, as opposed to locating it on the resolver, a gateway, or a LAN forwarder.

Table 1. **Client Side**

Integrated, native HTTP-based stub in browsers
On-host standalone DNS-HTTP forwarder
On-LAN DNS-HTTP forwarder
Enterprise-wide DNS-HTTP gateway
Upgrade to the host stub resolver
Drop-in replacement for stub

■ Server Side: Topological Options

We have identified a non-exhaustive list of ways that the server side subsystem of DJSON could be incorporated into the DNS resolution process. Broadly, the set of locations where the server side could be implemented are: 1) on the resolver, 2) on a stand-alone web server, or 3) as a dedicated gateway. In addition to the primary topological role, additional HTTP and DNS features can be added. These include: 1) DNSSEC validation (on behalf of non-DNSSEC-capable clients), 2) DDoS mitigation systems/devices preventing spoofed source IP traffic, 3) white-listed client gateways (protects against non-spoofed volumetric attacks), 4) TLS/SSL encryption, 5) cryptographically-signed JSON, 6) Load-balanced front-end, and/or 7) Globally distributed "anycast" service.

In our implementation we tested DJSON against all three main design options, but the most interesting results were obtained in stand-alone web server and dedicated gateway.

Implementation

In order to better understand the implementation of DJSON, we break it down into several elements:

- Client connection handling

- Server connection handling
- JSON encoding/decoding

A variety of client and server implementations representing each topological location were developed and tested. The objective in implementing DJSON is to preserve binary DNS packets across a round-trip, as illustrated in Fig. 1. For DNS queries, the translations are DNS-JSON on the client followed by JSON-DNS on the server. For DNS replies, the translations are DNS-JSON on the server followed by JSON-DNS on the client. In both directions, an identical before-and-after packet payload comparison is used for validation.

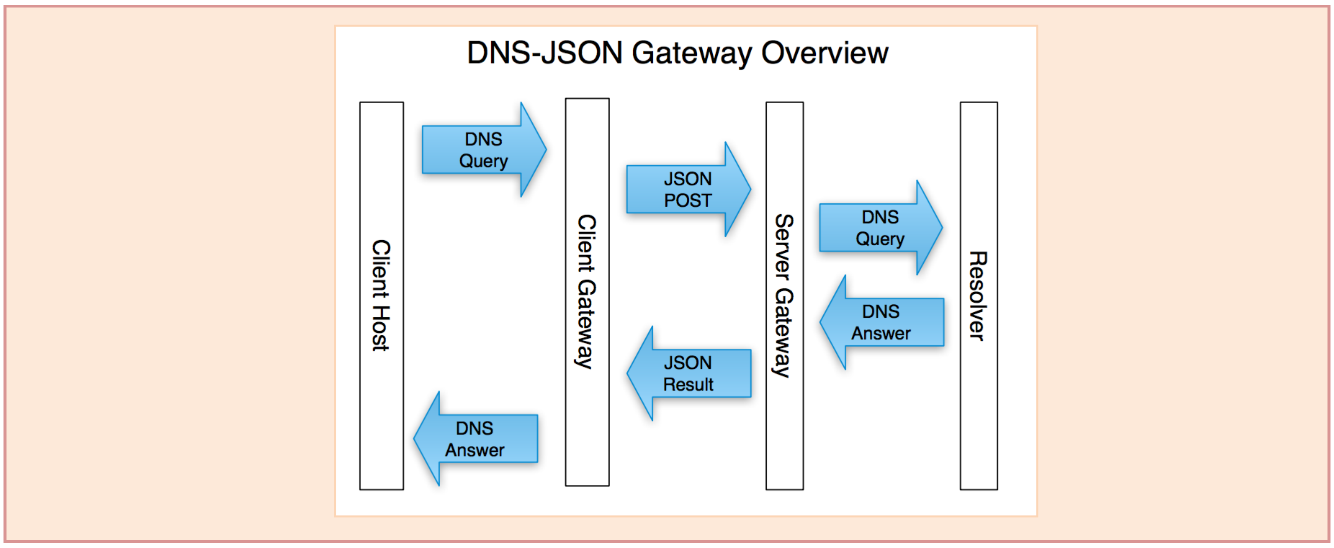

Figure 1. Overview of packet-preserving gateway function in both directions

Client Connection Handling

Regardless of where it is located, the current implementation of the client always behaves the same. The client attempts to keep a connection open to its configured server, over HTTP. It waits until a new DNS packet arrives. It parses the DNS packet, constructs the corresponding JSON message, and sends that to the server in an HTTP POST command. It waits for the response, parses the JSON, and builds an appropriate DNS response packet, and sends it to the originating host. When necessary, it will re-open the connection to the server on the HTTP TCP port.

Server Connection Handling

There are two server implementations which are as follows.

CGI module installed on a web server: This is called by the web server with the JSON payload provided. The program is executed for every packet, and exits after processing a single packet.

Stand-alone program: This needs to perform all webserver functions. It can maintain a persistent connection, and continue to execute (or sleep) between packets. This version has less overhead, and a much faster response to queries.

In both cases, the server converts the JSON payload to DNS wire format, and attempts to contact the configured DNS resolver. If it fails to receive an answer, it retries a preconfigured number of times, sending an error message back to the client if it never succeeds. If at any point it succeeds, the resulting answer packet is converted into JSON and sent back to the client.

JSON Encoding/Decoding Scheme

JSON was selected for the encoding scheme for a variety of reasons. JSON is well defined and supports bidirectional encoding/decoding for all atomic data types. JSON is widely supported, which is good given the number of configurations identified for this.

A descriptive/prescriptive representation of the structure, field data types, and parameterizations for the current state of the DNS standards was used. Then, the atomic data type encode/decode rules were created. Thirdly, a Label Dictionary mechanism was created to track label compression. The result is a deterministic method for encoding and decoding DNS packets.

DNS Message Format Definition: This **consists** of field definitions of the fixed portions of the DNS wire format, and the structure and relationship of variable components. It defines the atomic elements, names, types, and count-based quantifiers, such as the count of resource records per section, or the length of certain fields.

Resource Record Type Definitions: Similar to the overall Message Format, each resource record has its particular structure defined, including atomic type and size references. All current (non-obsolete) codes, options, flags, and types are implemented, with two exceptions: meta-types AXFR and IXFR. These are currently defined in a small number of tables. However, given that IANA publishes the list of names and types in XML format [12], one possible improvement would be to have the software detect changes in the IANA table (e.g. via periodic queries), and update itself as the table changes.

Validation and Measurement

The implementations of both the CGI and standalone server were tested with instances of both on-host, on-resolver, and stand-alone clients. The same sets of queries were performed across all combinations of client/server, as well as directly via several known good DNS resolvers. Unit testing of the perl module's encode/decode functions were conducted to confirm the source(s) of the observed levels of latency. The measurements reported below are from over 100 testing cases (averaged or represented as box plots to capture the distribution).

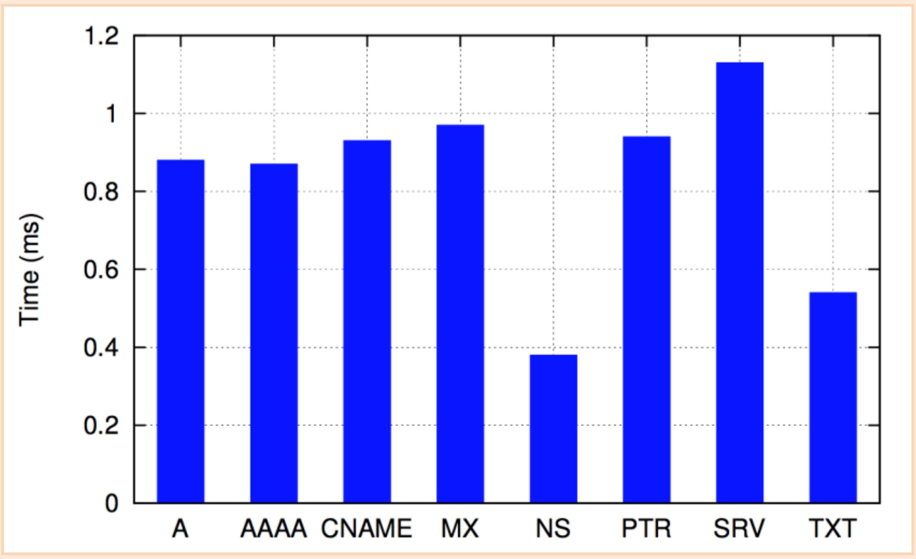

Figure 2. Typical query/answer encoding/decoding times, which are applied on a round-trip four times (one encode, one decode, on each query and answer packet)

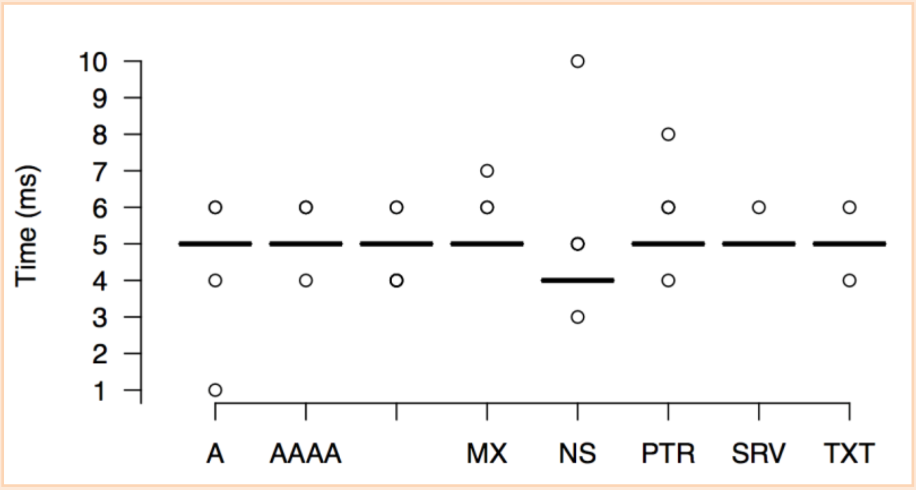

Figure 3. Latency when using a local DJSON gateway running as a dedicated listener (instead of being called via a generic web server)

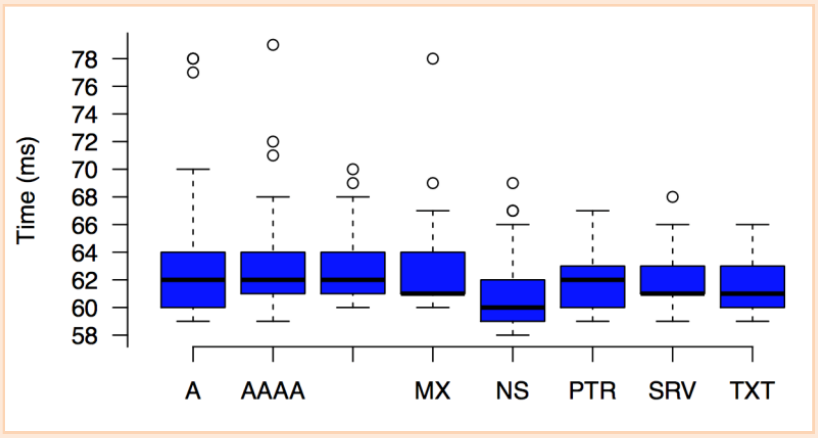

Figure 4. Latency when Apache2 calls the local DNS gateway as a CGI-BIN Perl script for every packet

The graph in Fig. 2 shows the latency of JSON encoding/decoding. The graph in Fig. 3 shows latency for queries to a CGI server, while Fig. 4 shows the latency for resolution when using a non-CGI dedicated server, with the server, client, and resolver all being on the same machine. A similar comparison was done using a remote DNS server, for both direct queries (in Fig. 5) and CGI-server queries (in Fig. 6). All queries were performed using the “dig” tool included with ISC BIND9, and the resulting packet formats were validated both by “dig” and via inspection using “tcpdump.”

In contrast, the latency when directly querying a local resolver should be near zero, since no networking or encoding/decoding is involved.

The test results were better than expected for a proof-of-concept implementation. Improvements in performance may be needed for Enterprise-level gateways, and can be achieved with very modest changes to the implementation, such as using threads to make use of multiple CPU cores on the client and server gateways. Additional improvements by implementing the gateway software in a compiled language like “C” rather than an interpreted language (perl), or by Web-server modules like “mod perl” are likely to improve the performance on both latency and overall system capacity (for query load and number of stub clients). The testing results show that a DJSON implementation would be practical to deploy and use with no detrimental impact to user experience. Experiments with a demonstration of DJSON confirm the lab results, where users were unaware that DJSON was in use.

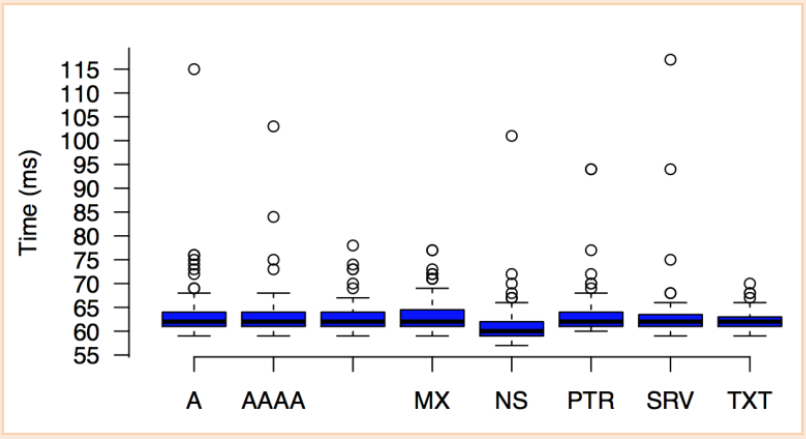

Figure 5. Latency when no DJSON gateway is used while querying a remote DNS server

Discussion

The test results demonstrate the feasibility of deploying DJSON in a large variety of real-world scenarios. The current proof-of-concept has been successful in achieving its stated goals. The modest latency overhead is comparable or less than the variance in resolution of real-world domain names. In addition, this latency is not compounded by the serialization of queries, since most host systems and applications (e.g. browsers) perform DNS pre-fetching and parallelize DNS resolutions with multiple threads.

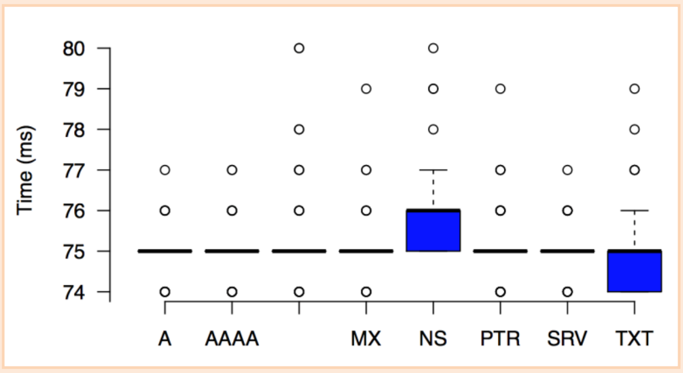

Figure 6. Latency when a DJSON gateway forwards to a remote DNS server

Enabling DNSSEC Validation

The original specifications for DNS [14], [16], [17] provided limited protection against off-axis threats, and were open to MitM attacks. DNSSEC [7]–[9], [13] provides cryptographic protection to DNS content. However, until DNSSEC upgrades are done on end hosts, they continue to be vulnerable to attack. DJSON provides a means by which end hosts gain the benefit of DNSSEC without having to upgrade their local software. This is accomplished by simply configuring the DJSON gateway to use a DNSSEC-validating resolver.

High-level Security Issues

One measure of systemic security risk is the attack surface. While JSON transport reduces several risks, the question that needs to be considered is, has any new attack surface been introduced?

There is a small set of Denial of Service vectors that are introduced, in terms of the pairs of gateways and the connections between them. That risk can be mitigated on the UDP side of gateways with Access Control Lists (ACLs), and with DDoS protection/mitigation techniques/services on the TCP side of the gateways. Additionally, client gateways can be "white-listed".

Additional Security and Privacy Capabilities

The fact that HTTP transport is utilized creates additional opportunities for security and privacy, via TLS. Transport Layer Security (TLS/SSL) is the end-to-end encryption used in HTTPS connections. By providing DJSON as a service over an HTTPS protected web server, the DNS traffic automatically becomes encrypted. Digitally signing, either with signed JSON or with TSIG in the DNS payload, prevents modification at a modest performance penalty. This feature may help address some practical and timely operational issues [22]

Conclusion and Future Work

DJSON has been demonstrated to be practical to deploy. DJSON servers do not perform heavyweight DNS resolution, so individual gateways can scale very well. The desired attributes detailed in Section II (highly supported protocol, no host upgrades, rewrite-hardened, censor-resistant) are met handily. DJSON is uniquely designed to meet these goals, as we showed in Section IV. The performance of DJSON has been shown to be deterministic, and thus scales linearly with load. DJSON is flexible; gateways can be placed anywhere, with many-to-one client-server gateways supported, and with client gateways supporting multiple stub resolvers.

This work is anticipated to become the starting point for a variety of related works. This includes integration into web browsers and lightweight smartphone/tablet apps, flexibility for far-end resolver choice, and additional implementations.

References

[1] http://www.bortzmeyer.org/dns-lg.htm.

[2] JSON-DNS. https://github.com/jpf/jsondns, October 2012.

[3] Dns in client-side javascript. http://www.fileformat.info/tool/rest/dns-json.htm, July 2013.

[4] Multilocation dns looking glass. http://www.dns-lg.com/, July 2013.

[5] REST-DNS Project. urlhttp://restdns.net/, July 2013.

[6] Anonymous, "The collateral damage of internet censorship by dns injection," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 3, pp. 21–27, Jun. 2012. Article (CrossRef Link)

[7] R. Arends, R. Austein, M. Larson, D. Massey, S. Rose. DNS Security Introduction and Requirements. RFC 4033 (Proposed Standard), Mar. 2005. Updated by RFCs 6014, 6840.

[8] R. Arends, R. Austein, M. Larson, D. Massey, S. Rose. Protocol Modifications for the DNS Security Extensions. RFC 4035 (Proposed Standard), Mar. 2005. Updated by RFCs 4470, 6014, 6840.

[9] R. Arends, R. Austein, M. Larson, D. Massey, S. Rose. Resource Records for the DNS Security Extensions. RFC 4034 (Proposed Standard), Mar. 2005. Updated by RFCs 4470, 6014, 6840, 6944.

[10] T. Berners-Lee, R. Fielding, H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945 (Informational), May 1996.

[11] S. Crocker. Host Software. RFC 1, Apr. 1969.

[12] IETF. Domain name system (dns) parameters. http://bit.ly/1bZ08qY, July 2013.

[13] B. Laurie, G. Sisson, R. Arends, D. Blacka. DNS Security (DNSSEC) Hashed Authenticated Denial of Existence. RFC 5155 (Proposed Standard), Mar. 2008. Updated by RFCs 6840, 6944.

[14] M. Lottor. Domain Administrators Operations Guide. RFC 1033, Nov. 1987.

[15] P. Mockapetris. Domain names: Implementation specification. RFC 883, Nov. 1983. Obsoleted by RFCs 1034, 1035, updated by RFC 973.

[16] P. Mockapetris. Domain names - concepts and facilities. RFC 1034 (INTERNET STANDARD), Nov. 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592, 5936.

[17] P. Mockapetris. Domain names - implementation and specification. RFC 1035 (INTERNET STANDARD), Nov. 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604.

[18] NLnet Labs. DNSSEC Trigger. http://www.nlnetlabs.nl/projects/dnssec-trigger/.

[19] L. Popa, A. Ghodsi, I. Stoica, "Http as the narrow waist of the future internet," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pp. 6, ACM, 2010. Aricle (CrossRef Link)

[20] M. Walfish, J. Stribling, M. N. Krohn, H. Balakrishnan, R. Morris, S. Shenker, "Middleboxes no longer considered harmful," in *OSDI*, vol. 4, pp. 15–15, 2004.

[21] N. Weaver, C. Kreibich, V. Paxson, "Redirecting dns for ads and profit," in *USENIX Workshop on Free and Open Communications on the Internet (FOCI),* San Francisco, CA, USA (August 2011), 2011.

[22] A. Mohaisen, A. Mankin. Evaluation of privacy for DNS private exchange. IETF Internet Draft, https://tools.ietf.org/html/draft-am-dprive-eval-01, July 2015.

Aziz Mohaisen obtained his M.S. and Ph.D. in Computer Science from the University of Minnesota, both in 2012. He is currently an Assistant Professor at the Computer Science and Engineering Department of the State University of New York at Buffalo. From 2012 to 2015, he was a Senior Research Scientist at Verisign Labs. Before pursuing graduate studies at Minnesota, he was a Member of the Engineering Staff at the Electronics and Telecommunication Research Institute, a large research and development institute in South Korea. His research interests are in the areas of networked systems, systems security, data privacy, and measurements. Dr. Mohaisen is a Senior Member of the Institute of Electrical and Electronics Engineers (IEEE) and Association for Computing Machinery (ACM).

Manar Mohaisen received his M.S. in communications and signal processing from the University of Nice-Sophia Antiplois, France, in 2005 and Ph.D. from Inha University, Korea, in 2010, both in communications engineering. From 2001 to 2004, he was with the Palestinian Telecom Co., where he was a cell planning engineer. Since Sept. 2010, he is with the Department of EEC Engineering, KoreaTech, Korea, where he is an assistant professor. His research interests include 3GPP LTE/-A systems, MIMO detection, and precoding and social networks.