

**CHARACTERIZATION OF DNS SERVERS FOR LATENCY  
ESTIMATION METRICS**

An Undergraduate Research Scholars Thesis

by

**JOSEPH JOHNSON**

Submitted to the Undergraduate Research Scholars program at  
Texas A&M University  
in partial fulfillment of the requirements for the designation as an

**UNDERGRADUATE RESEARCH SCHOLAR**

Approved by Research Advisor:

Dr. Dmitri Loguinov

May 2017

Major: Computer Science

# TABLE OF CONTENTS

	Page
ABSTRACT .....	1
NOMENCLATURE .....	2
SECTIONS	
I. INTRODUCTION .....	3
II. THE TURBO KING ALGORITHM .....	5
Algorithm Description .....	5
Turbo King Shortcomings .....	7
Objectives .....	10
III. FINDINGS .....	11
Forwarders .....	11
Varying Behavior between DNS Versions .....	13
Results of Internet Scan .....	14
IPv6 Findings .....	16
IV. FUTURE PROSPECTS .....	18
V. CONCLUSION .....	20
REFERENCES .....	21

# ABSTRACT

Characterization of DNS Servers for Latency Estimation Metrics

Joseph Johnson  
Department of Computer Science and Engineering  
Texas A&M University

Research Advisor: Dr. Dmitri Loguinov  
Department of Computer Science and Engineering  
Texas A&M University

It is easy to “ping” a remote computer from one’s own to determine the time delay between them, but estimating the latency between two remote hosts is much less trivial. This problem has various applications: social networking and online gaming are a few significant examples. Clever algorithms have been developed to approach the problem, though recent changes in the Internet are rendering them increasingly inaccurate. One such algorithm, *Turbo King*, makes use of DNS servers geographically near two remote hosts to estimate the latency between them. Unfortunately, with the emergence of IPv6, certain DNS server implementations have added additional queries and steps to their name resolution process. Because these extra delays are unpredictable, they add error to Turbo King’s latency estimation, rendering the algorithm imprecise. We have reevaluated Turbo King by classifying DNS servers by behavior and analyzing delays for potential patterns. With this information, we have patched certain cases to allow Turbo King to work correctly; in other cases we have been able to exclude IPs from the set of those usable by Turbo King. Now the algorithm’s range of usefulness has been expanded, possibly with enough coverage to measure the entire web.

## NOMENCLATURE

DNS	The Domain Name System (DNS) is used on the Internet to resolve hostnames (such as <code>www.example.com</code> ) into IP addresses.
Nameserver	One of many servers across the Internet which resolve hostnames. Every domain has one or more authoritative nameservers.
Query	A request issued to a nameserver. For example, when a user tries to visit <code>google.com</code> from a web browser, a DNS query is issued to resolve <code>google.com</code> .
King, Turbo King	Algorithms leveraging DNS to estimate the time delay between two hosts on the Internet.
IPv6	A newer version of the Internet Protocol which allows for longer, higher-resolution IP addresses.
A, AAAA	These query types request an IP address for a hostname. Type A asks for IPv4, the shorter addresses which are most familiar, while AAAA requests ask for IPv6.
Forwarder	When one nameserver does not know the answer to a query, it may ask another server on its behalf. This second server is known as a forwarder.

# SECTION I

## INTRODUCTION

Estimation of distance and latency between remote hosts on the Internet has become a prominent topic over the last several years. This problem has a variety of applications, such as closest-server selection, but relatively few algorithms can accurately perform it. IDMaps [3], GNP [4], and King [2] are a few examples of tools to predict the latency between two remote hosts. The first two methods require expensive deployment of infrastructure (e.g., tracer servers) or explicit cooperation of the remote hosts, whereas King leverages the Domain Name System (DNS), which does not impose these restrictions [3], [4]. King uses an observation that end-users are often topographically close to their authoritative nameservers. Instructing one DNS server to perform a lookup in another DNS server allows King to estimate the delay between them, which in turn approximates the delay between the clients. Unlike standard DNS, which is interested in resolving valid names, the King process is centered on measuring the time it takes for the two DNS servers to communicate.

Turbo King [1] improves upon the original King algorithm by utilizing an existing list of nameservers compiled from over the world, thus reducing the number of queries necessary for each measurement. Another improvement Turbo King has over the original is detection and avoidance of *forwarders*, which are hidden servers that perform lookups on behalf of other nameservers. Presence of forwarders skews the delay measurement and renders King inaccurate. However, despite these additional features, Turbo King has certain limitations that were brought to light in the past three years by the recent changes to the Internet (i.e., deployment of IPv6). Newer versions of the most popular DNS implementations, such as Microsoft DNS or BIND,

have changed their functionality to accommodate the evolving Internet and in doing so have introduced new delays when responding to DNS queries. Turbo King is unable to cancel out (or estimate) these delays using end-to-end measurement and thus suffers a significant accuracy loss. In order to make the tool effective again, it is necessary either to identify and use only a subset of DNS servers, or to patch Turbo King such that it can extract the correct result when using problematic servers.

## SECTION II

### THE TURBO KING ALGORITHM

In order to understand the problems facing Turbo King, it is necessary to be familiar with the steps of the algorithm. The first phase of our research focused on identifying why some servers do not work with Turbo King. We were able to fix some of these cases by modifying the original Turbo King algorithm to introduce additional measurements and heuristics.

#### Algorithm Description

There are four key components in a single run of the original Turbo King algorithm: the client and three DNS servers. We refer to the client running a Turbo King implementation as “C”. The two servers between which we are trying to measure latency are known as “X” and “Y”. A third, highly specialized DNS server known as “S” runs along-side C and is discussed in detail below. Figure 1 shows the high level path which DNS packets take starting from and ending at C, completing a total of 6 steps:

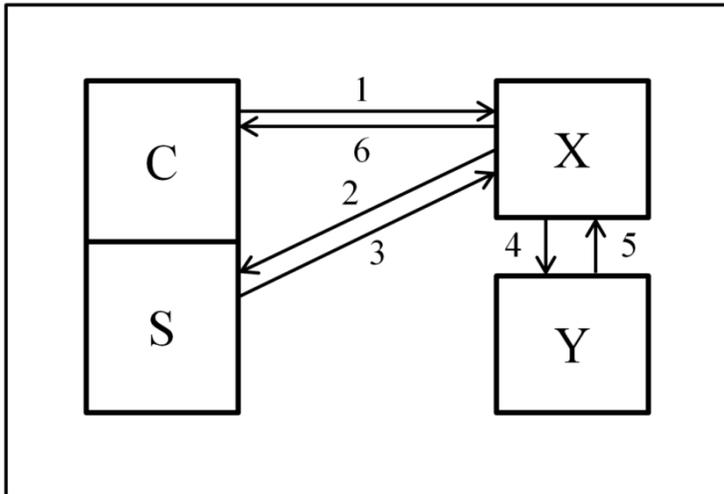


Figure 1 – Illustration of the six steps of the Turbo King algorithm.

The details of each step are as follows:

1. C sends a query “Q” to server X. Q consists of a random variation on a hostname within a domain we own. Since we can control which nameservers are authoritative for our own domain, we can control where X visits next in step 2 as it tries to resolve our question.
2. According to our setting with the domain registrar, X is pointed back to the computer that issued the initial query. A scaled-down DNS server “S” runs within the Turbo King program strictly to handle this request. Note that X must have recursion enabled as a DNS server property in order to be usable in Turbo King, otherwise it would not be able to visit S.
3. Instead of returning the actual IP address for Q (which is nonexistent anyway), we tell X to search again at nameserver Y.
4. X asks Y to resolve Q. This is the first component of the goal value, the round-trip delay between X and Y.
5. Y responds to X with an error because it cannot find Q.
6. X forwards that error back to our client.

After step 6, we have all the information we need to extract the round-trip delay between X and Y. It is not possible to compute time T of steps 4 and 5 directly because we do not have access to X or Y, but we can calculate it by subtraction:  $T = t_{36} - t_{12}$  where the subscripts indicate between which steps the time is recorded [1].

## **Turbo King Shortcomings**

This elegant algorithm once functioned well, but changes to DNS servers and the structure of the Internet have introduced several cases where Turbo King fails. Through our findings we have identified three main scenarios that cause failure: the presence of DNS forwarders, differences in the behavior of nameserver versions, and IPv6 prevalence.

### *DNS Forwarders*

As stated above, server X must be recursive in order to receive our request and consult our specialized server S. However, there is no guarantee that X will not ask other servers instead of S, thus taking additional time. Ultimately, S will be queried as it is the only authoritative server for our domain, but there potentially could exist a chain of forwarding servers between X and S.

The presence of these X-forwarders is handled in Turbo King by the assumption that any forwarder used by X is sufficiently nearby to X that we can safely assign X to the forwarder and get a very similar time [1]. Not just any nameserver is a candidate for X, so if the forwarder is

not recursive, the algorithm still fails. We shall discuss in a later section a new method of eliminating forwarders while keeping the original X.

Furthermore, the Turbo King paper offers no solution for forwarders encountered from server Y. Y does not have to be recursive, indeed we would prefer that it respond immediately with anything, even an error. However, there is a chance that Y could attempt to resolve Q recursively in the same manner as X, ultimately causing a cycle. Forwarders are just one example of how Turbo King fails to cover all possibilities to obtain a correct result.

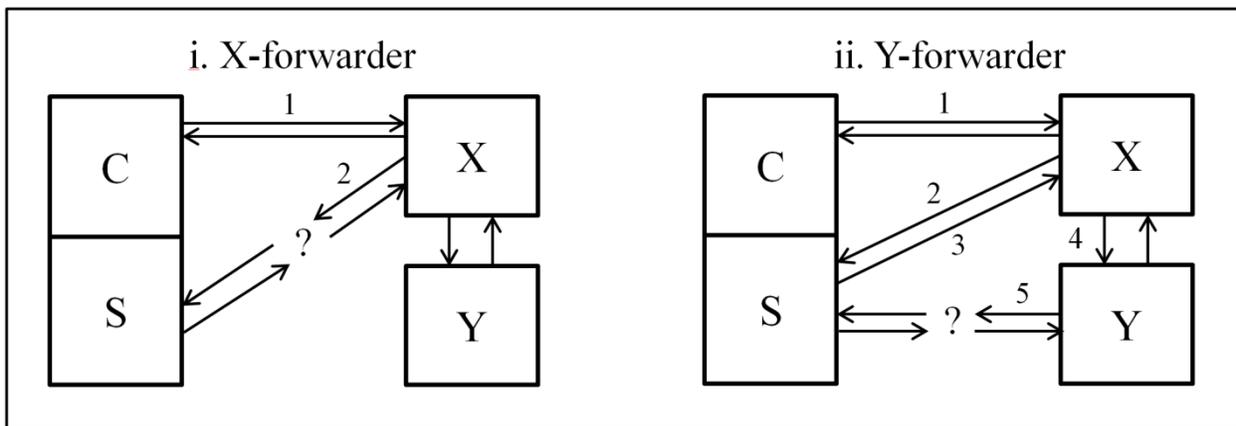


Figure 2 – The presence of forwarders makes Turbo King unable to calculate an accurate estimate.

### *Differing Behavior between DNS Versions*

Although there are well-defined standards for DNS servers to follow, administrators still have significant leeway regarding how to handle requests (e.g. which error codes to use, how to react to queries for certain domains, and so forth). Also, older servers may not understand requests which make use of new DNS features, causing their responses to be unpredictable.

Within the last couple of years, BIND began using Extended DNS (EDNS) fields as part of its standard functionality. BIND uses EDNS to send security signatures [6], but an older

server might fail to correctly parse a packet with such a signature. This can be problematic for Turbo King, as in the case where a new version of BIND is used for X and an older server for Y. EDNS fields will be sent along with Q to Y, but Y could respond in a number of different ways depending on its type and version. One example we have seen is the error code FORMERR, indicating a format error. X usually retries the query in this case without EDNS, requiring twice the original time before X finally comes back to our client C.

Regardless of EDNS, Y's behavior still varies greatly depending on its version. As we discussed in the previous section, Y might try to resolve Q through forwarders. Another option besides a forwarder is one of many error codes, some of which prompt X to resend the query and add more delay. Y may also elect to refer X to Top Level Domain (TLD) nameservers, which are used to resolve requests for their designated TLD (such as ".com"). Altogether, it is naïve to assume that all servers behave identically when used as Y; different responses could cause one or more retries from X, all of which increase the error in Turbo King's time estimation.

### *IPv6 and Evolving Internet*

When Turbo King was first introduced, IPv6 was not as prevalent as it is today. We are now seeing DNS servers which, when acting as X, try volleys of additional IPv6 requests either before or after querying Y in Turbo King. Though Q was of type A (in DNS terminology, an IPv4 request), these servers desire to give us any answer other than an error, so they continue the search using AAAA (IPv6) requests. It is clear to see how this adds more delay to the measurement process. Original Turbo King did not consider IPv6 so AAAA queries are yet another case that makes Turbo King unusable.

## Objectives

Our primary objective in this research has been to modify the Turbo King algorithm to automatically detect and eliminate delay added to our measurements. We have examined the categories of problematic servers in order to either modify Turbo King to use them, or mark them as unusable. In the latter case, a subset of all DNS servers must be removed from the set appropriate for Turbo King. Fortunately, due to the sheer number of nameservers distributed all over the world, even a fraction of the total may still be sufficient to estimate latency between any desired end-points.

The challenge has been exhaustively determining which servers are absolutely unusable. Since one of the major issues with Turbo King is related to varying behavior between server versions, an intuitive first step is to eliminate servers running problematic versions. For example, there is no guarantee that a server we have categorized as using AAAA will use the same extra steps as another which uses AAAA. Furthermore, even a specific server might behave differently when responding to one instance of our algorithm versus another. All we can do in these cases is flag the servers involved as unusable since their behavior is unpredictable.

In summary, our objectives have been 1) to patch the Turbo King algorithm to handle cases in which it currently fails and 2) to eliminate servers causing failures for which we cannot fix the algorithm, thus creating a new list of completely usable servers. Together, these goals directly contribute to our overall objective of making Turbo King correct again.

## **SECTION III**

### **FINDINGS**

We have made a number of interesting findings and fixes during this project which relate directly to the difficulties listed above. We have found that we can programmatically solve the problem of forwarders as well as some cases of version disparity. Unfortunately, we have also identified sets of servers for which there does not appear to be a programmatic solution. The only option for these is to remove them from Turbo King's master list.

#### **Forwarders**

Recall that DNS forwarders pose a problem because we have no way of knowing the path and associated latency from the measurement server (X or Y) to S. If a forwarder is detected from X, the Turbo King paper recommends stopping the algorithm and retrying with the forwarder as X the next time, if it is recursive. We have developed another way to programmatically subtract out the time taken by forwarders so that it is not necessary to exclude any additional servers from Turbo King's list.

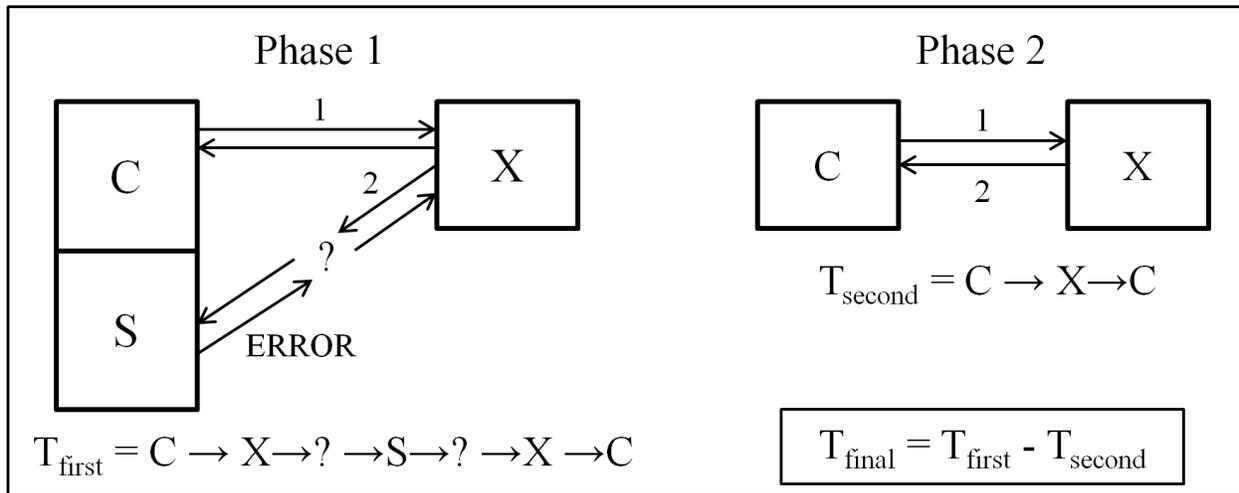


Figure 3 – A two-phase process to compute and remove delay caused by X-forwarders.

Our process involves running Turbo King as normal, but sending additional packets after its completion for the purpose of finding the time caused by forwarders. The proper value to subtract from Turbo King’s full run can be computed in two phases. Phase 1 begins like a regular instance of Turbo King, but when S receives a packet from X (through the forwarder), it responds with an error (such as NXDOMAIN, non-existent domain) instead of Y’s location. Measuring this process gives us a nearly correct time; we need only eliminate the round trip from C to X. This is trivially done with a cached query or server status request in Phase 2. The resulting time as computed in Figure 3 represents forwarder delay and can simply be subtracted from the original Turbo King value. It is important to note that X does not use the same forwarder every time and some may be slower or faster than others. Redundant iterations are necessary to filter these possibilities and improve accuracy. The amount of runs is arbitrary and should be adjustable depending on network constraints or algorithm speed requirements.

Regarding Y-forwarders, which the Turbo King paper does not address, there is a simpler way to solve cases in which they are an issue. Due to the purpose of Y in the structure of the algorithm, there is no reason we should not cache an answer for Q in Y prior to starting Turbo

King. Telling Y that Q does not exist (via the NXDOMAIN code) beforehand will guarantee in some cases that Y will not try to perform additional searching. It will simply respond immediately with the cached answer, skipping forwarders altogether.

A reader with knowledge of DNS might ask at this point how it is possible that Y-forwarders exist. By specification, the recursive X server asks Y on our behalf in a non-recursive manner (to prevent excess depth in the tree of servers for a given exchange). Therefore, Y should not offload this request to a forwarder since it is an iterative request. However, not all machines which respond to DNS requests are full-fledged DNS servers. Simple modems and other such devices might only be able to forward a DNS request to a real server. Thus, the quick and easy caching method will not work for all Y servers. However, we can still apply the algorithm discussed above for X-forwarders and obtain a correct result.

### **Varying Behavior between DNS Versions**

The most interesting findings we have made in this category deal with the EDNS issue as well as a seemingly unsolvable Windows DNS problem.

Regarding EDNS fields sent from X in step 4, we have found that Y servers will either 1) handle the request because they support EDNS or 2) respond with some kind of error because they cannot handle EDNS. X servers will receive this error and react depending on what type of error code it is. Since BIND uses EDNS, we tested the reaction of BIND for commonly seen error codes and in every case the correct answer can be determined through repeated runs, as BIND caches that EDNS is not supported in Y. We simply take the minimum time from running Turbo King back-to-back. Other Y behavior unrelated to EDNS proves to be more difficult. If

X receives a list of Top Level Domain or Root servers from Y, there appears to be no fixed reaction: sometimes X will ask Y a second time, other times it will return to C as desired. Thus, many different types and versions of DNS servers are unusable for Turbo King.

## **Results of Internet Scan**

In February 2017, we performed scans of the Internet to compile an updated list of global DNS servers. This process involves crawling the IP address space (excluding certain ranges which are restricted or special-purpose), sending DNS packets to simulate a Turbo King run, and waiting for a reply. Obviously, a very small percentage of all possible IP addresses are DNS servers, but the numbers we obtained were generally optimistic for measurement applications.

The space we scanned included 2.8 billion candidate IP addresses on the Internet. For each of these, we sent one iterative query and one recursive query, both of a similar form to Turbo King. The name we asked for was composed of a randomized hash and also the hexadecimal form of the IP being queried. The goal was for the iterative queries to simulate X to Y communication, while the recursive packets behaved like Turbo King (C to X). When S received a packet during, it would check the incoming IP address against the IP encoded within the query. This allowed us to determine which servers used forwarders.

Additionally, by returning the encoded IP as an authoritative answer, we aimed to identify servers which always returned an incorrect answer. Previous studies have shown that not all DNS servers respond truthfully for reasons such as parked websites and even governmental restrictions [7]. Servers returning TLD or Root servers also count as incorrect in our characterization since they neither performed our lookup nor responded with an error.

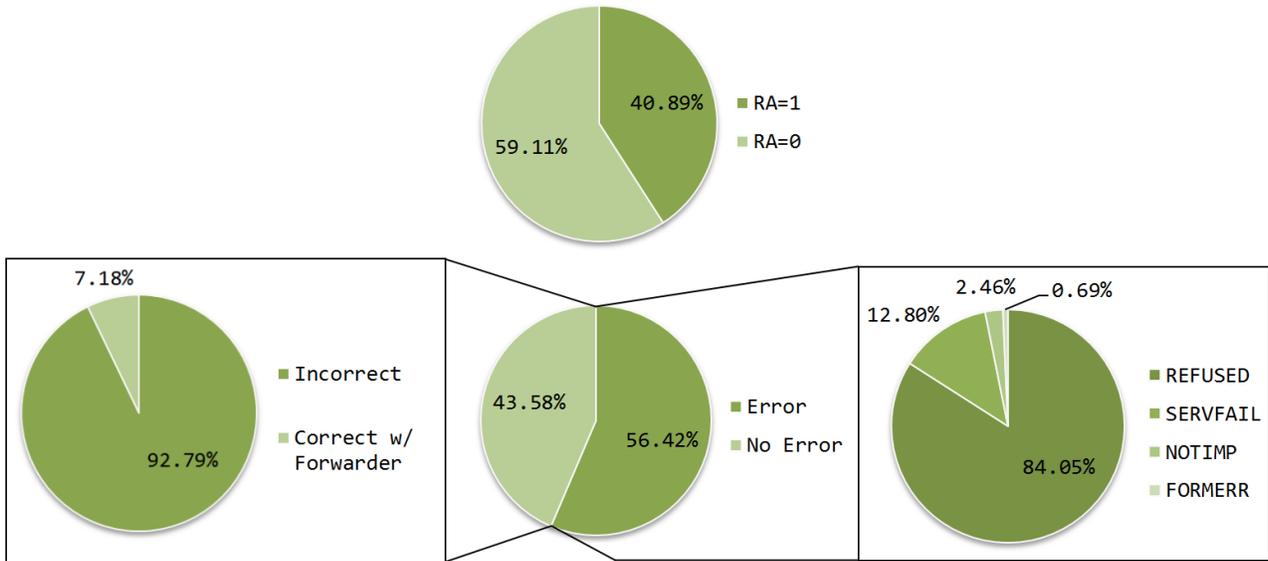


Figure 4 – **Iterative** breakdown. 8,187,358 unique IPs responded to DNS queries during the scan. The top-middle pie chart shows Recursion Available (RA) statistics. The bottom charts show a breakdown by response received.



Figure 5 – **Recursive** breakdown. 8,080,911 unique IPs responded to DNS queries during the scan. The top-middle pie chart shows Recursion Available (RA) statistics. The bottom charts show a breakdown by response received.

As discussed in a previous section, servers with Recursion Available can be used as X in the Turbo King algorithm, whereas Y does not need to support recursion. Thus according to these results, at most about half of these servers are candidates for X (not yet counting further

breakdowns). Also about half of the total is the amount which responded with the NOERROR code. Compared to previous knowledge, it is very striking that we identified forwarders in nearly 90% of these NOERROR servers. Fortunately, the methods we have developed to eliminate forwarders mean that these cases are still open for use by Turbo King. Regarding the other NOERRORs, about 12% responded incorrectly (either lying as mentioned in [7] and above, or with TLDs/Roots), and only 1522 servers on the whole Internet behaved as correct, open resolvers with no forwarder. Examining the responses to iterative queries (the green graphs), 56% of servers responded with an error, overwhelmingly REFUSED. We believe that these cases can be used as Y (some may also be modems with Y-forwarders). Again, the only condition for a suitable Y is that it responds with *anything* immediately, with no extra steps.

Other scans performed by our lab in the past have produced more DNS responding servers (max 22M, this time we had only 8M unique IPs), so future scans will help refine the list. By percentage, however, we are satisfied with these results in terms of making Turbo King effective on the net.

## **IPv6 Findings**

After scanning the entire Internet, we began parsing the IP lists we obtained into smaller categories. We developed a program called XTester which reads input IPs and simulates Turbo King, but with a local Y server which we control. We used XTester to search for trends among the servers which supported recursion and returned a correct result, with or without the use of forwarders. Due to the volatility of open resolvers around the world, results differ depending on the time of day we perform the X test, but we found a surprisingly large proportion of AAAA

behavior in these servers. A minimum of 75% and maximum of 85% were identified to send us AAAA queries of some kind. These results seem to parallel the high percentage of forwarders seen in the previous section; the Internet has truly evolved since Turbo King was first conceived.

In light of this discovery, the question is how many of these IPv6-supporting servers can be used for Turbo King. Further testing needs to be done, but from taking investigating various IPs from the list it appears most AAAA noise occurs before Step 3 of Turbo King (the referral to Y). This is fortunate because we really only need perfectly predictable behavior during X's exchange with Y. The extra packets complicate things somewhat, but all we have to do is correctly identify and ignore them, a relatively small change to Turbo King overall. In general, even the highly prevalent IPv6 behavior looks optimistic for making Turbo King effective on the Internet once again.

## SECTION IV

### FUTURE PROSPECTS

There are a couple follow-up projects involving Turbo King which we would like to study in the future. Firstly, we want to produce a matrix containing calculated delays between any two points on the Internet. This will be a challenge in itself, but once completed, would be very useful. Our second follow-up project would use this matrix as a publicly-available delay lookup table. Anyone around the world could use this service to find Turbo King's result for virtually any two end points.

There are two major challenges that make generation of an exhaustive delay matrix difficult. Obviously, the dimensions of this matrix would be huge (as seen by our Internet scan) and would require unreasonable time and space to compute and store. However, we know that the IP space is divided into subnets (by BGP prefix) such that delays vary mainly among subnets [8]. To illustrate by example, if machines 1 and 2 are in subnet A and machines 3 and 4 are in subnet B, it is reasonable to guess that the time between 2 and 4 is roughly the same as that between 1 and 3. By this observation, we can pare down the dimensions of our matrix to include not every IP, but just one candidate IP from every subnet. This brings us to the second challenge: finding an appropriate, Turbo King-usable server in every BGP prefix. There are currently about 600K prefixes [9] on the web which are frequently changing, and it is currently unknown how many of these we can cover with usable X's and Y's for Turbo King.

Assuming we produce this matrix and cover all (or at least most) of the Internet, we want to make it easily available to the public. One possible system would be a web app, hosted by our

lab. A user could enter two IPs or domain names and the app would retrieve the data stored offline in our delay matrix. Perhaps once a month or so we could freshen the matrix with another Turbo King run using the most current BGP list. If successful, this would be the first publicly-available table of latencies between all points on the Internet.

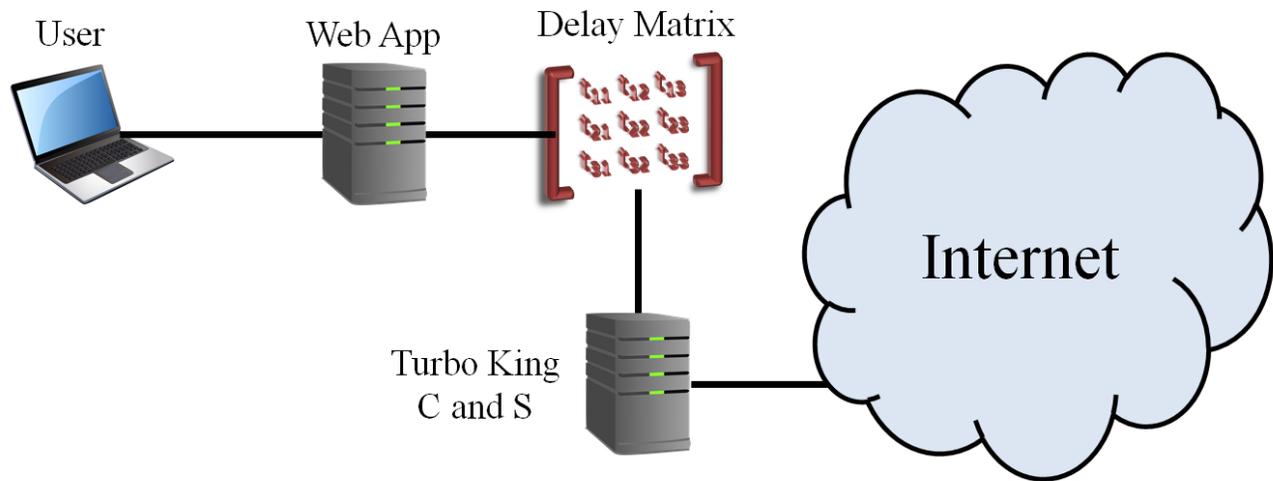


Figure 6 – Anyone would be able to access Turbo King data via our web app. The matrix is periodically updated by Turbo King runs.

## **SECTION V**

### **CONCLUSION**

Through experimentation and analysis of behaviors we have categorized many of the major issues facing the Turbo King algorithm. Some of these we have successfully addressed and patched; others we have determined to be unsolvable. We have also produced an updated master list of responding DNS servers across the entire IP address space. This huge list has been pared down into smaller sets of usable servers for X and Y. Because of our modifications to Turbo King which expand its usefulness and due to the sheer number of servers around the world, we hope Turbo King will be able to estimate latency between nearly any two points. We are optimistic that we will fully solve the problem and restore Turbo King to its former status as a useful Internet latency estimation tool.

## REFERENCES

- [1] D. Leonard and D. Loguinov, “Turbo King: Framework for Large-Scale Internet Delay Measurements,” in *IEEE INFOCOM*, Apr. 2008.
- [2] K. P. Gummadi, S. Saroiu, and S. D. Gribble, “King: Estimating Latency between Arbitrary Internet End Hosts,” in *Proc. ACM IMW*, Nov. 2002.
- [3] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. “IDMAPS: A Global Internet Host Distance Estimation Service,” in *IEEE/ACM Trans. on Networking*, Oct. 2001.
- [4] E. Ng and H. Zhang. “Predicting Internet Network Distance with Coordinates-Based Approaches,” in *Proc. IEEE INFOCOM 2002*, New York, NY, Jun. 2002.
- [5] V. Gite, “Find out DNS Server Version With DNS Server Fingerprinting Tool,” in *nixCraft*, 2007. [Online]. Available: <http://www.cyberciti.biz/tips/howto-remotely-determine-dns-server-version.html>. Accessed: Sep. 13, 2016.
- [6] C. Almond, “Refinements to EDNS fallback behavior can cause different outcomes in Recursive Servers,” in *ISC Knowledge Base*, 2014. [Online]. Available: <https://deephought.isc.org/article/AA-01219/0/Refinements-to-EDNS-fallback-behavior-can-cause-different-outcomes-in-Recursive-Servers.html>. Accessed: Jan. 22, 2017.
- [7] M. Kühner, T. Hupperich, J. Bushart, C. Rossow, and T. Holz. “Going Wild: Large-Scale Classification of Open DNS Resolvers,” in *ACM IMC '15*, Tokyo, Japan, Oct. 2015.
- [8] Y. Rekhter, T. Li, and S. Hares, “A Border Gateway Protocol 4 (BGP-4),” in *RFC 4271*, 2006. [Online]. Available: <https://tools.ietf.org/html/rfc4271>. Accessed: Apr. 8, 2017.
- [9] T Bates, P. Smith, and G. Huston, “CIDR REPORT for 8 Apr 17,” in *CIDR Report*, Apr. 8, 2017. [Online]. Available: <http://www.cidr-report.org/as2.0/>. Accessed: Apr. 8, 2017.