Authority Server Selection of DNS Caching Resolvers

Yingdi Yu
UCLA
yingdi@cs.ucla.edu

Duane Wessels
Verisign
dwessels@verisign.com

Matt Larson
Verisign
mlarson@verisign.com

Lixia Zhang
UCLA
lixia@cs.ucla.edu

ABSTRACT

Operators of high-profile DNS zones utilize multiple authority servers for performance and robustness. We conducted a series of trace-driven measurements to understand how current caching resolver implementations distribute queries among a set of authority servers. Our results reveal areas for improvement in the “apparently sound” server selection schemes used by some popular implementations. In some cases, the selection schemes lead to sub-optimal behavior of caching resolvers, e.g. sending a significant amount of queries to unresponsive servers. We believe that most of these issues are caused by careless implementations, such as keeping decreasing a server’s SRTT after the server has been selected, treating unresponsive servers as responsive ones, and using constant SRTT decaying factor. For the problems identified in this work, we recommended corresponding solutions.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Performance, Measurement, Domain Name System

Keywords

DNS, Server Selection

1. INTRODUCTION

The Domain Name System is a fascinating, versatile, and fundamental component of Internet communications. It is highly distributed both in ownership and operation, used both as a target and as a vehicle of numerous types of attacks, and generally a very resilient protocol. The DNS resolution plays a significant role in the Internet-user experience. It is often the first step in establishing a communication channel between sender and receiver. If DNS resolution is slow or not working, we take notice.

This work is focused on the behavior of *caching resolvers*, whose role is to receive queries from end user applications, which are then resolved by querying some number of authority servers. This process is also known as *iteration*[1]. As the name implies, caching resolvers store received answers for future use. Cache hits obviously improve performance, and have been intensively studied [4, 10, 18].

Another way that caching resolvers improve performance is by minimizing the iterative query delay. Many zone operators use multiple authority servers for both performance and robustness. If the resolver can successfully predict which one will respond in the shortest time, the delays experienced by users can be minimized. However, simply sending all queries to the least-latent authority server may not be the best policy. A resolver with predictable behavior is more susceptible to Kaminsky-style cache poisoning [8].

The DNS specifications [13] are vague on server selection algorithms (“Find the best servers to ask” and “Get the answer as quickly as possible”). However, there seems to be consensus among implementors and users that, for a given zone, a caching resolver should prefer the least-latent authority server (for good response time) yet still query the others (to distribute the load and monitor their performance). Therefore, the first question we try to answer in this paper is: *Does an implementation prefer the least-latent authority server?*

We find some implementations that do not always prefer the least-latent authority server. Some of them, as will be discussed in Section 4.1, spread out queries evenly among all authority servers. Others, which intend to send queries to the least-latent server, fail to distribute queries as expected. This implies that some defects exist in server selection schemes. Therefore, the second question we want to answer is: *What are those defects that make some implementations prefer servers with longer latency?*

Thus far our questions assume an unchanging network. In reality, networks are dynamic. For example, a change in routing may increase the Round-Trip Time (RTT) to a server, or queries to a server go unanswered for some amount of time. Caching resolvers that can efficiently detect negative changes can avoid performance degradation. Meanwhile, caching resolvers should also be able to detect positive changes, such as when a server begins responding again, or when latency to a server has decreased. It is generally more difficult to detect positive changes, because larger-latent servers are queried less frequently. Therefore, the third question we try to answer is: *Does an implementation detect network changes, especially positive changes, in a timely manner?*

To answer these three questions, we measured the server selection schemes of six popular caching resolver implementations, including BIND 9.7 and 9.8, PowerDNS, Unbound, dnscache and Windows DNS, under a number of different

[1] This process is also called “recursion” sometimes.

(simulated) network scenarios. The six implementations adopt different server selection schemes and behave differently. Some behaviors are sub-optimal, either because the resolver sends more queries to larger-latent (or even unresponsive) servers, or because positive network changes are not detected quickly. In studying the server selection schemes, we identified three factors responsible for these behaviors.

Contributions of our work are follows:

- We measured server selection schemes of six popular caching resolver implementations, which we believe represents the majority resolver software currently in use [3]. To the best of our knowledge, this paper is the first to study DNS performance from the perspective of authority server selection.
- We found four types of sub-optimal server selection behaviors.
- We identified three factors responsible for these sub-optimal behaviors, thus providing guidelines for future work in server selection schemes. Although our work is focused on DNS, these guidelines are also applicable to other systems where server selection is necessary.

The rest of the paper is organized as follows. Section 2 presents background information on authority server selection. We describe the measurement testbed in Section 3. In Section 4, we study the four types of sub-optimal server selection behaviors. Related works are listed in Section 5, and we conclude in Section 6.

2. AUTHORITY SERVER SELECTION

When a caching resolver needs to query a domain, it may want to select one of the authority servers serving the domain for queries. Some caching resolvers may select the least-latent authority server in order to minimize the query delay. Some others may randomly pick a server, so that the query load can be balanced on all authority servers. Moreover, random selection can make the behavior of these caching resolvers less predictable, thus preventing these resolvers from malicious attacks, such as Kaminsky-style cache poisoning [8]. Note that even those caching resolvers which select the least-latent server may still have to periodically query the rest servers, in order to tell which server has the least RTT when network is dynamically changing.

A caching resolvers usually selects an authority server in two steps. The first step is to estimate the RTT of each authority server[2] based on statistics of previous queries. (If a query times out, the value of timeout timer or a large preset value is taken as the RTT.) In order to mitigate RTT fluctuation, Smoothed RTT (SRTT), a weighted moving average of previous queries' RTTs, is used as the estimated RTT.

The second step is to select an authority server according to the estimated RTT. Here, two types of selection schemes can be applied. The first one, which we call *Least SRTT Selection*, is to select the server with the least SRTT. Using this scheme alone may be problematic. For example, when the RTT of a server changes from a large value to the minimum one, the server can be selected only after its estimated RTT is updated. However, in order to get the latest RTT, the server has to be queried at first. As a result of deadlock, this server may be never selected. Therefore, SRTT decaying is used to force a caching resolver to poll each authority server periodically. SRTT decaying is a mechanism that decreases the SRTT of unselected servers by a factor $\beta < 1$ when a response arrives.

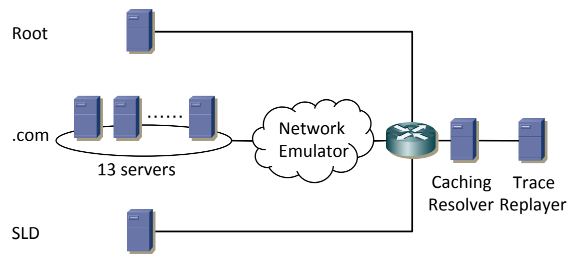


Figure 1: Testbed structure. The trace replayer sends out requests of name lookups. Some lookups may trigger the caching resolver to iteratively query the DNS infrastructure. Queries to the TLD servers experience delay and packet loss created by network emulator.

The decaying factor β can be constant or exponentially-decreasing. For example, BIND [2] uses a constant $\beta = 0.98$, while PowerDNS [1] adopts an exponentially-decreasing one:

$$\beta_n = e^{\frac{t_n - t_{n+1}}{C}} \tag{1}$$

where t_n and t_{n+1} are timestamps of two consecutive responses, and C is a constant.

A constant β couples SRTT decaying and the iterative query rate. SRTT decays more quickly when the iterative query rate is higher. An exponentially-decreasing β, however, decouples SRTT decaying from the query rate. Consider, for example, two query traces covering the same amount of time (t_1, t_2). One contains two queries which arrive at t_1, t_2 respectively, while the other one contains three queries which arrive at t_1, t', t_2. The decaying ratio of the first trace is

$$e^{\frac{t_1 - t_2}{C}}.$$

For the second trace, it is

$$e^{\frac{t_1 - t'}{C}} \cdot e^{\frac{t' - t_2}{C}} = e^{\frac{t_1 - t_2}{C}}.$$

Decaying ratios of both traces are equivalent, and are only determined by constant C.

The other selection scheme is called *Statistical Selection*. This scheme selects a server with a probability which is a function of the server's SRTT. In this way, even the server with the largest SRTT can be selected for some queries, though its being-selected probability may be very small.

3. MEASUREMENT TESTBED

Our measurement is a trace-driven study, and is conducted in an isolated environment. As shown in Figure 1, our testbed consists of four components: a DNS infrastructure, a network emulator, a caching resolver, and a program that replays lookup traces.

Authority server selection is only meaningful when servers support the same domain. In our testbed, ".com" is chosen as the target domain for the convenience of collecting a large

[2] Without explicit note, authority servers in this paper serve the same domain.

Scenario ID	Description
Scenario 1	RTTs of authority servers range linearly (10ms) from the first one (50ms) to the last one (170ms).
Scenario 2	A variant of Scenario 1, where the first authority server is unresponsive (100% packet loss ratio).
Scenario 3	A variant of Scenario 2, where the unresponsive authority server recovers after 5 minutes.

Table 1: Measurement Scenarios

enough DNS lookup trace. The tested ".com" domain is served by 13 authority servers. A root zone server is setup to respond with referrals to the ".com" servers. Another server is setup with wildcards [13], so that it can serve all the Second Level Domains (SLDs) under ".com", and terminates all DNS lookups. In order to eliminate differences in measurement results caused by cache ejection and refresh in measurement, the TTL of all DNS resource records and the size of the resolver's cache are set to large enough values. We have to point out that shorter TTL and smaller cache only lead to more overall queries, but do not directly affect server selection.

A network emulator that can adjust packet delay and loss ratio is placed between the caching resolver and the ".com" servers. Using this emulator, we model several network scenarios to measure different aspects of server selection. In this paper we discuss the three scenarios listed in Table 1. The purpose of Scenario 1 is to measure how a caching resolver distributes queries among authority servers with different RTTs. The purpose of Scenario 2 is to see whether a caching resolver can detect an unresponsive server and avoid querying it. The purpose of Scenario 3 is to measure how long it takes a resolver to detect the recovery of an unresponsive server and select it again.

The DNS lookup trace is from a 10-minute log of a resolver of a large U.S. ISP. It consists of approximately 3.5 million lookups for 408,808 unique DNS names (all ending with ".com") belonging to 154,165 SLDs. Assuming no packets are lost and all DNS records stay in cache once they have been fetched, the average rate at which a caching resolver queries the tested ".com" domain is about 250 queries per second.

We measure six widely used implementations of caching resolvers[3] as listed in Table 2. We capture iterative query traces at the caching resolver side. All of the analysis in this paper is based on these captured traces.

4. MEASUREMENT RESULTS

We find four types of sub-optimal server selection behavior in the captured query traces. Two of them are found in Scenario 1: RTT-insensitive server selection (i.e., selecting a server regardless of its RTT) and RTT-proportional server selection (i.e., sending more queries to larger-latent servers). Another type is found in Scenario 2: Unresponsive server blackhole (i.e., sending a significant amount of queries to unresponsive servers). The last type is found in Scenario 3: Slow reaction to positive changes (e.g., a server recovers from unresponsive status). In this section, we discuss the consequences and explain the reasons for each type of behavior.

Caching Resolver	Version
BIND	9.8.0, 9.7.3
PowerDNS	3.1.5
Unbound	1.4.10
DNSCache	1.05
WindowsDNS	6.1 (Windows Server 2008)

Table 2: Caching resolvers measured and their versions

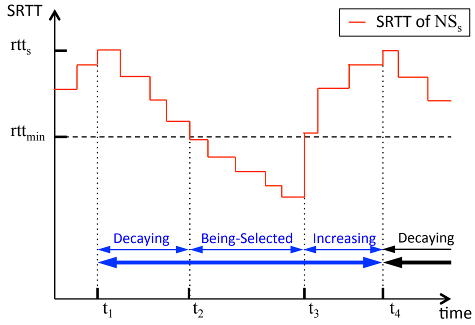


Figure 3: The SRTT variation of an example server NS_e in BIND 9.8. The four solid bars represent four queries to NS_e. NS_e is selected when its SRTT is less than $rrt_{\min}$.

4.1 RTT-Insensitive Server Selection

We find that three implementations (DNSCache, Unbound, and Windows DNS) distribute queries evenly among all the authority servers. Two reasons can account for this behavior. First, some caching resolvers do not estimate the RTT of authority servers (e.g., DNSCache). Second, some caching resolvers only use the estimated RTT to rule out underqualified servers, rather than to select the least-latent server. For example, Unbound randomly selects among servers with the SRTT less than 400ms.

Due to sending a considerable amount of queries to larger-latent servers, the RTT-insensitive server selection scheme inevitably increases average query delay. This can become more obvious if most of the servers have larger RTTs. And such situations are not rare on the Internet.

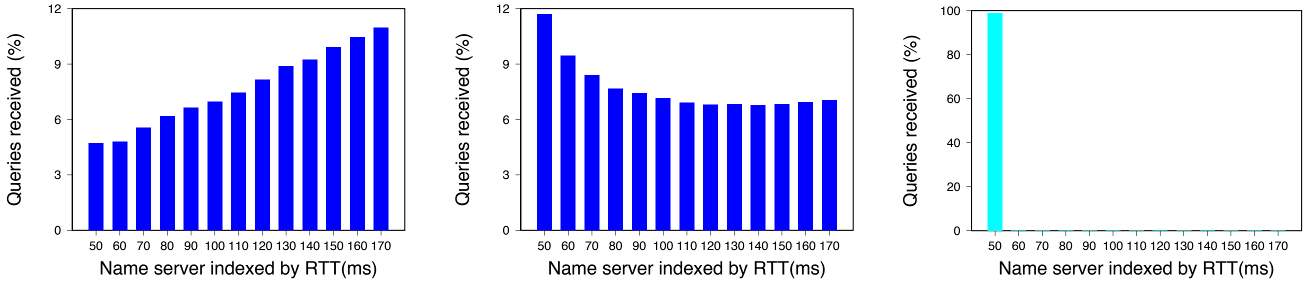
4.2 RTT-Proportional Server Selection

We find that BIND 9.8 send more queries to larger-latent authority servers in Scenario 1[4] as shown in Figure 2a. We call this type of behavior as RTT-proportional server selection. However, BIND 9.8 is implemented with the Least-SRTT Selection scheme, and should select the server with the least SRTT, yet our data indicates otherwise. To understand the reasons for this behavior, we investigate the process of server selection inside BIND 9.8.

Consider a domain that is served by n authority servers. We use one of them, NS_e $(1 \leq e \leq n)$, as an example to describe how an authority server is selected by BIND 9.8. Fig-

[3] Nominum DNS was unavailable to us, and was not measured.

[4] ISC has been notified about this issue.



(a) (b) (c)

Figure 2: Query distributions in Scenario 1: (a) BIND 9.8; (b) BIND 9.8 with 1/10 query rate; (c) PowerDNS.

ure 3 illustrates the SRTT variation of NS$_e$ in BIND 9.8. As can be seen, such variation is periodical, and can be divided into three phases: *decaying*, *being-selected* and *increasing*.

A decaying phase starts when the last response from NS$_e$ is received by BIND 9.8 at t_1. In this phase, since NS$_e$ is not selected, all received responses are from the other servers. As a result of SRTT decaying, these responses make the SRTT of NS$_e$ decrease. Eventually, the SRTT of NS$_e$ becomes the minimum one at t_2, and the decaying phase ends.

The being-selected phase is next to the decaying phase. In this phase, although NS$_e$ has been selected, some responses to queries sent before t_2 are still in flight. When these outstanding responses are received by BIND 9.8, the SRTT of NS$_e$ continues to decrease until the first response from NS$_e$ is received. The being-selected phase ends at t_3 when the SRTT of NS$_e$ is no longer the minimum one. As the SRTT of NS$_e$ is always the minimum one in the being-selected phase, all the queries are sent to it in this phase. If NS$_e$ has a large RTT, its SRTT may not be the minimum one after having been updated by the first response from NS$_e$. Therefore, the being-selected phase usually lasts for only one RTT.

The increasing phase is next to the being-selected phase. In this phase, all the received responses are from NS$_e$, except those repsonding to queries sent after t_3. The SRTT of NS$_e$ may continue increasing until t_4 when the last response from NS$_e$ is received, which also indicates the beginning of the next decaying phase.

Note that NS$_e$ can be selected only in the being-selected phase during an SRTT variation period, and that once NS$_e$ has been selected it continues being selected for one RTT. Therefore, the percentage of queries sent to NS$_e$ is:

$$p_{\text{NS}_e} = \frac{P_e \cdot t_{sel_e}}{\sum_{i=1}^{n}(P_i \cdot t_{sel_i})} \tag{2}$$

where t_{sel_i} is the length of the being-selected phase of authority server NS$_i$, and P_i is the probability with which the SRTT of NS$_i$ is the minimum one. When decaying phase is long enough, P_i can be approximated as:

$$P_i \approx \frac{t_{sel_i}}{t_{dec_i} + t_{sel_i} + t_{inc_i}} \tag{3}$$

where t_{dec_i} denotes the length of the decaying phase of authority server NS$_i$, and t_{inc_i} for the length of the server's increasing phase.

An authority server with a long RTT usually has a long decaying phase. According to (3), such a server can be selected with a small probability. Although this server may be queried for a long RTT, the total percentage of queries sent to it is still small according to (2).

If the decaying phase is very short (e.g., when SRTT decays very quickly), formula (3) can not reflect the being-selected probability accurately any more. However, this formula still provides some useful implication. Note that both of t_{sel} and t_{inc} are approximately equal to the server's RTT. When t_{sel_i} approaches to zero, P_i in formula (3) is approximately equal to 0.5. This implies that all the authority servers have almost the same probability to become the one with the minimum SRTT if SRTT decays very quickly. Assume that each authority server has the same P_i, an RTT-proportional query distribution can be obtained from (2).

In order to validate our speculation that RTT-proportional server selection is caused by fast SRTT decaying, we conduct another experiment. As we mentioned in Section 2, if a constant decaying factor is used, SRTT decays slower when the iterative query rate is lower. Given BIND uses a constant decaying factor ($\beta = 0.98$), we slow SRTT decaying by replaying DNS lookup trace at 1/10 of the original speed, and measure the query distribution again. As shown in Figure 2b, although many queries are still sent to large-latent servers, the least-latent server now receives the most queries.

As a result of coupling SRTT decaying and the iterative query rate, the RTT-proportional server selection most probably happens when a popular domain is queried by a caching resolver that serves many end users. Although only a few domain-resolver pairs satisfy this condition, the number of the affected end users and the importance of the affected domains should not be underestimated.

An exponential decreasing factor, used by PowerDNS, can decouple SRTT decaying and the average query rate. With a large constant C in (1), SRTT decays very slowly in PowerDNS (it takes 42 seconds for SRTT to decay to a half). As a result, more than 99% of queries are sent to the least-latent server, as represented by the spike in Figure 2c. However, we have to point out that slow SRTT decaying does not avoid sending queries to large-latent server for one RTT periodicaly.

Unlike caching resolvers discussed above, BIND 9.7 selects an authority server based on an SRTT-related probability. As a result, no authority servers can be continuously selected for a long time. The query distribution in BIND 9.7 in Figure 4 can reflect to some degree the difference in authority

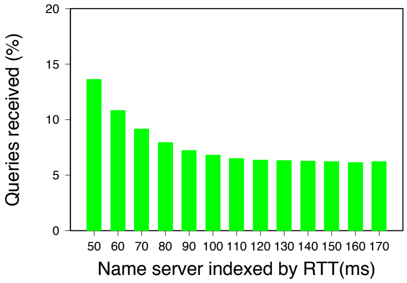


Figure 4: Query distribution of BIND 9.7 in Scenario 1

servers' RTTs, However, as SRTT decaying is still used in BIND 9.7, the SRTT of a large-latent server is less than its real RTT if the iterative query rate is higher. Therefore, a considerable amount of queries are still sent to large-latent servers. As can be seen in Figure 4, all servers receive at least 7% of queries.

4.3 Unresponsive Server Blackhole

We find that two implementations (BIND and DNSCache) send a significant number of queries to the unresponsive server in Scenario 2. A queriy sent to an unresponsive server times out in the end, and forces a caching resolver to re-send the query to other servers. As a result, the delay of the corresponding DNS lookup increases significantly. DNSCache does not keep statistics of previous queries, thus failing to avoid selecting unresponsive server. Therefore our analysis is focused on BIND in this section.

As shown in Figure 5, BIND 9.8 sends 23% of queries to the unresponsive server. BIND 9.7, whose result is not shown here, sent 11% of queries to the unresponsive server. The reason for this behavior is that BIND takes a timed out query as a "responded" query with an extremely long RTT which is equal to the value of timeout timer. According to DNS specification [13, 14], the timeout timer should be set to at least four seconds. Within such a long "RTT", an unresponsive server can force BIND to send a huge amount of queries to it.

A desirable authority server selection should avoid such an "unresponsive server blackhole". Two other implementations (Unbound and WindowsDNS) achieve this goal. Unbound handles unresponsive and responsive servers separately. When a server is detected as unresponsive, Unbound periodically sends a query to probe the server until it becomes responsive again.

The rest caching resolver, PowerDNS, decreases SRTT very slowly as mentioned in Section 4.2. An unresponsive server has to wait a very long time (3 minutes) to be selected. Therefore, only a few queries are sent to the unresponsive server. However, as an unresponsive server is still taken as responsive, once it is selected, PowerDNS will send all following queries to it till the timeout timer expires. As a result, queries to a domain with an unresponsive server may periodically experience large delay.

4.4 Slow Reaction to Positive Changes

We list in Table 3 the time that each implementation needs to detect the recovery of the unresponsive server. Among them, PowerDNS and Unbound take a long time to detect the server's recovery. It may be acceptable that a caching resolver fails to quickly detect the recovery of a large-latent server, because most of queries are still sent to the least-latent one. But if a caching resolver takes a long time to detect the recovery of the least-latent server, most of queries will be still sent to a sub-optimal server.

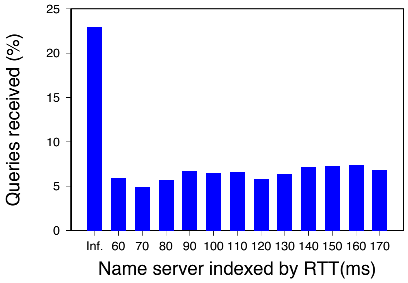


Figure 5: Query distribution of BIND 9.8 in Scenario 2. The first authority server is unresponsive.

We attribute the slow reaction to the frequency of querying (or probing) an unresponsive server. Unbound sets a 15-minute interval between two consecutive probes. This implies that, in the worst case, it takes up to 15 minutes for Unbound to detect the recovery of an unresponsive server.

For implementations that use least SRTT selection and SRTT decaying, a slow SRTT decaying implies a long waiting time to probe a server again. For example, PowerDNS has to wait three minutes to re-query an unresponsive server.

Although the least SRTT selection discussed above may react slowly to the recovery of an unresponsive server, the SRTT of the server has already decayed to a small value when it is selected again. In contrast, the performance of statistical selection may be worse, because the SRTT of the selected server may be still large when it is selected again. Therefore, statistical selection cannot frequently query a server with a large SRTT multiple times in a short time, it may take a longer time to decrease the SRTT of the server to a reasonable value.

A comprehensive comparison among different implementations is listed in Table 3.

5. RELATED WORK

Many previous efforts on server selection were focused on web servers [5, 7, 9, 16, 19]. However, methods presented by some work included bandwidth as one of selection criteria [5, 7, 9, 16]. Unlike web servers, communication between caching resolver and authority servers consists of short queries, therefore response time plays a more important role in DNS server selection. Some other work used methods unsuitable for DNS, such as application-layer anycasting [19], or proactive probing [16].

Server selection schemes in Content Delivery Networks (CDN) have also been studied intensively [11, 12, 15, 17]. However, server selection in CDNs is based on DNS. Authority servers of CDN networks use the IP addresses of caching resolvers to infer geolocation of users, and respond with IP addresses of the closest server. As being strongly coupled with DNS, these schemes are not applicable to server selection in DNS.

Studies on DNS specific server selection are limited. Deb et al. [6] proposed a server selection algorithm which im-

Implementation	Query distribution	Avoid querying unresponsive server	Time to detect server recovery
BIND 9.8	Proportional to RTT (if query rate is high); Inverse proportional to RTT (if query rate is low)	No	< 1 second
BIND 9.7	Inverse proportional to RTT	No	< 1 second
PowerDNS	Spike distribution	No	3 minutes
Unbound	Uniform distribution (among all authority servers with RTT < 400ms)	Yes	15 minutes
DNSCache	Uniform distribution	No	< 1 second
WindowsDNS	Uniform distribution	Yes	1 second

Table 3: Comparison of measurement results of caching resolvers on 1) Query distribution in Scenario 1, 2) Avoid querying unresponsive server in Scenario 2, and 3) Time to detect server recovery in Scenario 3.

proves the accuracy of SRTT estimation. Their proposal assumed that SRTT decaying can work as expected, but our study shows that this assumption may not be true in some situations. Ager et al. [4] studied the delay of DNS lookups from two perspectives: distance between end users and caching resolvers, and the efficiency of caching. In contrast, our work is focused on the delay between caching resolvers and authoritative authority servers.

6. CONCLUSION

In this work, we studied how widely used DNS caching resolvers implement authority server selection. It is the first study that compares different implementations. We found three issues inside some “apparently sound” server selection schemes. First, SRTT decaying, which is used to periodically probe an authority server, should stop once the server has been selected. Otherwise, the lasting SRTT decaying can force a caching resolver to query the server until the first response is received. Second, constant decaying factor couples SRTT decaying and the iterative query rate, and should be avoided. Otherwise, a large-latent server can be selected with a larger probability when query rate is higher. Last but not the least important, unresponsive authority servers should not be handled as responsive servers with a large RTT. Otherwise, once an unresponsive server is selected, all queries will be sent to this server until the timeout timer expires. Moreover, an unresponsive server should be fixed as soon as possible, because it may still significantly affect the performance of a domain even though the other authority servers work well.

7. REFERENCES

[1] PowerDNS. http://www.powerdns.com.

[2] BIND. http://www.isc.org/software/bind.

[3] DNS Surveys. http://dns.measurement-factory.com/surveys/.

[4] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig. Comparing DNS resolvers in the wild. In *ACM IMC*, 2010.

[5] R. Carter and M. Crovella. Server selection using dynamic path characterization in wide-area networks. In *IEEE INFOCOM*, 1997.

[6] S. Deb, A. Srinivasan, and S. Pavan. An improved DNS server selection algorithm for faster lookups. In *IEEE COMSWARE*, 2008.

[7] S. Dykes, K. Robbins, and C. Jeffery. An empirical evaluation of client-side server selection algorithms. In *IEEE INFOCOM*, 2000.

[8] S. Friedl. An illustrated guide to the kaminsky dns vulnerability. *Uinxwiz. net Tech Tips, August*, 2008.

[9] K. Hanna, N. Natarajan, and B. Levine. Evaluation of a novel two-step server selection metric. In *IEEE ICNP*, 2001.

[10] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. DNS performance and the effectiveness of caching. *Networking, IEEE/ACM Transactions on*, 10(5):589–603, 2002.

[11] B. Krishnamurthy, C. Wills, and Y. Zhang. On the use and performance of content distribution networks. In *ACM SIGCOMM Workshop on Internet Measurement*, 2001.

[12] R. Krishnan, H. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao. Moving beyond end-to-end path information to optimize CDN performance. In *ACM IMC*, 2009.

[13] P. Mockapetris. Domain Names - Concepts and Facilities. RFC 1034 (Standard), Nov. 1987.

[14] P. Mockapetris. Domain Names - Implementation and Specification. RFC 1035 (Standard), Nov. 1987.

[15] I. Poese, B. Frank, B. Ager, G. Smaragdakis, and A. Feldmann. Improving content delivery using provider-aided distance information. In *ACM IMC*, 2010.

[16] M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek. Selection algorithms for replicated web servers. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):44–50, 1998.

[17] A. Shaikh, R. Tewari, and M. Agrawal. On the effectiveness of DNS-based server selection. In *IEEE INFOCOM*, 2001.

[18] P. Vixie. What DNS is not. *Communications of the ACM*, 52(12):53–47, 2009.

[19] E. Zegura, M. Ammar, Z. Fei, and S. Bhattacharjee. Application-layer anycasting: A server selection architecture and use in a replicated web service. *Networking, IEEE/ACM Transactions on*, 8(4):455–466, 2000.