Overcoming DNSSEC Performance Issues with DHT-based Architectures

Daniel Migault*, Stanislas Francfort*, Stéphane Sénécal*, Emmanuel Herbert† and Maryline Laurent†
* Francetelecom, firstname.lastname@orange.com
† Institut Mines-TELECOM, UMR CNRS 5157 SAMOVAR, firstname.lastname@it-sudparis.eu

***Abstract*—DNSSEC deployment for large Internet Service Provider (ISP) is an important issue. With the current architecture, migration of current DNS resolving platform requires 5 times more nodes. This paper introduces alternative architectures where the DNS traffic is split between the nodes of the platform according to the queried Fully Qualified Domain Names (FQDNs), rather than the IP addresses of the queries. We show that such type of architecture requires up to 30% less nodes. However, this load balancing technic results in a non uniform distribution of the resources among the nodes of the platform. Furthermore, operational teams are reluctant to modify the existing load balancing infrastructure. Thus, we investigate how pro-active caching over a Distributed Hash Table (DHT) protocol, can optimize the resources of an ISP operational DNSSEC resolving platform. We find out that it can reduce the number of nodes by 3.5.**

I. Introduction

DNS is the protocol that makes possible communications based on names, by binding an IP address to a Fully Qualified Domain Name (FQDN). As such, end users rely on a DNS resolving platform to determine the IP address of the target, which makes DNS resolution platform a critical element of the Internet.
In July 2008, Dan Kaminsky showed that DNS was sensible to cache poisoning attacks [1], and that the long term solution was DNSSEC [2]. DNSSEC resolutions cost a lot more than a DNS resolution, and experimental measurements [3] show that depending on the implementation, migration to DNSSEC for a resolving platform requires 169% to 500% more resources. A crucial problem faced by ISPs is that they manage large DNS Resolution Platforms of around 100 nodes, and migration from a 100 to a 500 node platform represents too many Operational And Management (OAM) issues. The purpose of this paper is to provide an architecture that would optimize the DNSSEC resolving Platform so that ISP can migrate from DNS to DNSSEC. In order to optimize the resources of the platform, we attempt to limit DNS(SEC) operations that require a lot of CPU. We focused two major operations : DNS cache lookups and DNSSEC Resolutions.

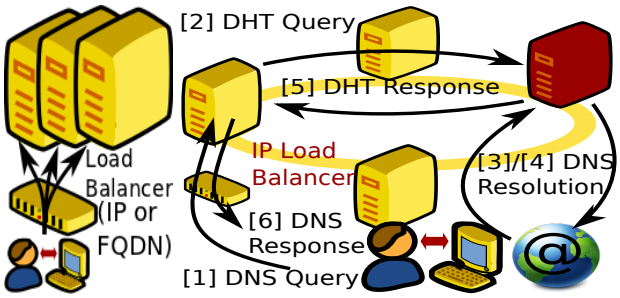
Current DNS Resolution platforms are called IP_{XOR} and are represented in figure 1a. They are composed of a load balancer [4] that splits the traffic among the nodes by XORing the IP addresses of the DNS query. Each node of the platform resolves the received DNS query. Such load balancer distributes uniformly the load between the nodes, but the same FQDN may be sent on different nodes, thus triggering parallel resolutions.

At first, we estimate how splitting the DNS(SEC) traffic between the nodes can significantly reduce the load of the platform. We use a load balancer that splits the DNS(SEC) traffic by hashing the FQDN. This architecture is called $FQDN_{SHA1}$ and is also represented in figure 1a. Contrary to IP_{XOR}, with $FQDN_{SHA1}$, the node the load balancer forwards the DNS query to is determined by the FQDN rather than the IP addresses. Simulations show that $FQDN_{SHA1}$ requires 30% less resources than IP_{XOR}. On the other hand, using such a load balancer, results in a very non uniform distribution of the resources among the nodes of the platform. More specifically, some nodes receive more queries or perform more resolutions than the others. As a result FQDN load balancing reduces the necessary resources, but the remaining challenge is making the resources uniformly distributed between the nodes.

A uniform distribution of the resources can be reached in two ways. One way is to define specific rules for the load balancer, that result in uniformly distributing the resources between the nodes. With this solution, all the intelligence is placed in the load balancer, and the nodes keep on responding to the queries that are sent to them. This alternative has been developed in [5], and made possible with [6] load balancers. In this paper, we adopt the opposite view. We leave the load balancer unchanged, like in IP_{XOR}, and require some intelligence in the nodes of the Resolving platform. With this architecture, the nodes cooperate with each other. We base the cooperation on Distributed Hash Table (DHT) mechanisms like Pastry. Our motivations for this architecture is mainly that load balancers are critical equipments under heavy load, which makes operational teams reluctant to any modification. Furthermore, modifying the nodes provides much flexibility for innovation, and we can benefit from multiple mechanisms that have been already elaborated. We evaluate the gains provided by the DHT proactive caching architecture described in [7], [8]. Modelisations and Simulations with our DNS traffic shows that we can easily improve the platform efficiency by 3.5 compared to the traditional architectures.

Note that in this paper, Pastry is used for a maximum of a few hundred nodes, while Pastry was originally designed for very large platforms of thousands or billions of nodes. As such, we do not consider the routing discovery mechanisms provided

by Pastry, but we take advantage of Pastry auto-configuration capacity, its robustness to Deny of Service (DoS), and its cache sharing mechanisms.
Note also that the optimizations discribed in this paper only consider DNSSEC. In other words, the gains provided by the proposed architecture do not apply for DNS. The reason is that DHT or cache sharing architectures balances the cost of a communication between nodes of the platform and a DNS(SEC) resolution. With DNS, resolution costs almost as much as a communication between nodes of the platform, thus reducing the advantage of the proposed architecture. On the other hand, DNSSEC resolution involves signature checks that cost much more than a communication between nodes of the platform. This provides significant advantage for cache sharing and DHT architectures.



(a) $FQDN_{SHA1}$, IP_{SHA1}, IP_{XOR} (b) DHT (Pastry)

Fig. 1. DNS Resolution Platform Architectures based on Load Balancers and on DHT

The paper is organized as follows. Section II positions our work. Section III presents simulations, with real DNS traffic, different types of load balancers that consider - like IP_{XOR} - the IP addresses of the DNS queries, or - like $FQDN_{SHA1}$ - the FQDN. $FQDN_{SHA1}$ is shown to be more efficient by up to 30%. Section III evaluates the gains provided by the DHT proactive caching architecture and shows that we can easily improve the platform efficiency by 3.5 compared to the traditional architectures. Section V concludes this paper.

II. Position of our Work

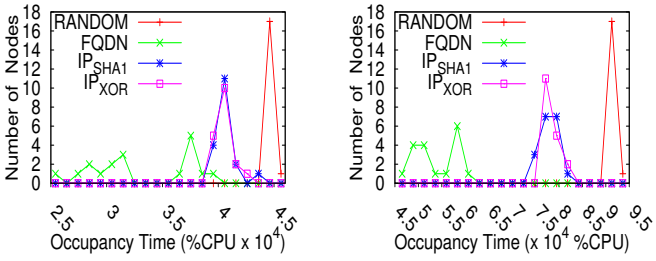
This paper studies how load balancing DNS queries to different nodes according to the queried FQDN rather than the IP addresses improves the efficiency of the platform. To deploy such a platform, we use the Pastry based architectures, and Pastry features which enhance the cache sharing mechanisms. As far as we know, no work has been conducted previously on the FQDN load balancing, however some has been done on DNS traffic analysis, resolving platform enhancement (cache and resolution), DNS platform running over DHT, as well as on Active Caching (AC).
[9] introduces the Zipf function for web traffic, and [10] applies it on DNS traffic. [10], [11] study how many end users should be aggregated to take advantage of the cache. [12], [13] describe CoDNS and how cooperation of multiple resolvers reduces the resolution's latency by relying on multilocated resolvers. Our work is focused on a local multinode platform, and thus all resolvers are expected to be located in a single place. Furthermore, our work is more concerned about resource optimization than resolution time.
[14] evaluates the performance of a DNS implementation over Chord [15] and DHash [16]. [17], [18] analyze how DHT enhances the Naming System robustness by considering *Data failure rate*, *Path failure rate* and *Path length*. DNS efficiency is related to the zone's popularity and label number, whereas DHT's efficiency is related to the RRsets's popularity. DHT main drawback is its heavy routing algorithms, however active caching - like Beehive [7] - achieves the same availability as the current DNS. [7], [8] show how Beehive takes advantage of a Zipf distribution to actively cache the proper data and achieves a $O(1)$ lookup performance over a DHT. DHT is also more robust to orchestrated attacks.
Our work differs from the above mentioned work on at least three aspects: we are using DHT protocols for a resolving platform, and not for authoritative servers. The difference between authoritative servers and resolving servers is that authoritative servers respond with data they are hosting. On the other hand resolving servers are requesting queries for resolution. Resolution may involve querying multiple authoritative servers, as well as cache management operations. Then, previous architectures did not consider DNSSEC. For authoritative servers, the difference between DNS and DNSSEC is mainly that larger data are hosted by the servers since signatures are added. However, for resolving server, DNSSEC makes resolution much more costly, since signature checks are required. At last, the scope of previous research work concerned the global Naming architecture, that is how all authoritative DNS servers could take advantage of a DHT architecture. In our case, DHT Pastry is only used for a local resolving platform. In other words, we are not considering a platform of more than 1000 independent nodes. Our work considers less than 200 nodes administrated by the same network administrator, on the same LAN.
However, our work is closely looking at the principle exposed in Beehive [7], [8] and takes advantage of the Zipf distribution of the DNS traffic. Simulations based on this principle shows that this pro-active caching mechanism makes our platform up to 3.5 times more efficient.

III. Load Balancing Architectures

This section evaluates the efficiency of different load balancing strategies from a 10 minute DNS live traffic captured on one of our 18 node clusters that composes the DNS Resolution platform. It shows that balancing traffic on the platform according to FQDN rather than IP address requires 30% less ressources.
To evaluate the various architectures, we compute the distribution of the CPU for each node of the platform. From the 10 minute DNS live capture, we replay the traffic and define the number of Cache Hit and Cache Miss on each node of the platform. To derive the global CPU of the node, we considered experimental measurements [3]. These measurements benchmark for various implementations and DNS(SEC) configurations the CPUs that correspond to a Cache Hit (CPU^H)

and a Cache Miss that requires a Resolution (CPU^R).
The studied load balancing strategies are based on IP_{XOR} and $FQDN_{SHA1}$ as mentioned in section I. We also introduce two other strategies: IP_{SHA1} where splitting is done over the SHA1 value of both IP source and destination of the queries, and *Random* where the node selection is randomly performed. IP_{SHA1} vs IP_{XOR} shows the efficiency of SHA1 vs XOR functions.

This section estimates CPU, the required CPU resources to handle with the traffic. We considered two distinct actions to be performed depending whether the FQDN is in the cache or not. When a cache Hit occurs, the resolving node performs a simple cache lookup and CPU resources are noted CPU^H. When a cache miss occurs, a resolution over the Internet is required, and the necessary resources are noted CPU^R. CPU is expressed with CPU^H and CPU^R by considering CPU^H and CPU^R (%CPU) for BIND9 ($0.015\,\%CPU$, $0.317\,\%CPU$). IP_{XOR} and IP_{SHA1} are very similar, and perform better than *Random*. $FQDN_{SHA1}$, on the other hand provides a bi-cluster distribution: the low CPU and high CPU groups. For DNS, the low CPU group has a high variance, and the mean CPU of the high CPU group almost equals CPU of IP_{SHA1}. With DNSSEC results are better, there are still two clusters but they have smaller variance, which means that the CPU is more uniformly distributed. The mean CPU value of both groups are closer to each other than in the case of DNS, and in any case much lower than with *Random* or IP_{SHA1} / IP_{XOR}.
$FQDN_{SHA1}$ seems promising since it offers a mean CPU lower than any of the other architectures. $FQDN_{SHA1}$ happens to be 1.117 more efficient for DNS, and 1.342 for DNSSEC. However its major drawback is that it presents a non uniform distribution. $FQDN_{SHA1}$ improves the platform's



(a) DNS - BIND9 (b) DNSSEC_SIG - BIND9

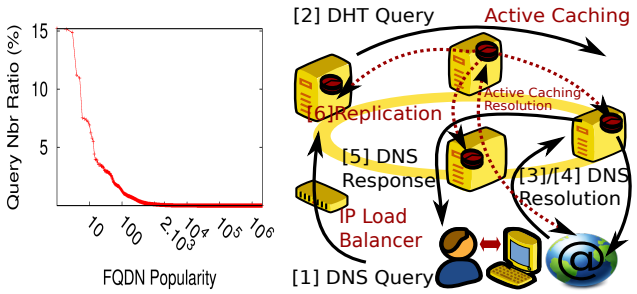
efficiency by roughly 30% compared to IP_{SHA1}. However, $FQDN_{SHA1}$ does not present a uniform distribution of the CPU among the nodes. This mostly results from the difference between the number of queries associated to FQDNs (the FQDN's popularity) and the number of queries associated to the users. FQDN's popularity presents a higher dispersion compared to the user's number of queries. In that sense, $FQDN_{SHA1}$ distribution can not rely on a hash function (SHA1) and we have to check that two popular FQDNs cannot be assigned to the same *Home*. Establishing such a distribution is out of scope of the paper, however we have several solutions in mind. For instance, for a given hash function, we can use a salt, concatenate the salt to the FQDN to be hashed and consider $hash(FQDN||salt)$. The salt is incremented until it splits properly the most popular FQDNs. Another alternative is to build a routing table for the most requested FQDNs [5], [19] whereas others are load balanced with the hash function.

IV. Pro-active Caching DHT Architecture

Section III shows that load balancing DNS traffic is more efficient with $FQDN_{SHA1}$ than with $IP_{SHA1/XOR}$. However, as shown in section I, load balancers that split the traffic according to the FQDN are not widely available and would require ISP to modify their current core infrastructure. Furthermore, on an architecture point of view, such load balancer still consists of a single point of failure. In order to apply $FQDN_{SHA1}$ principle, that is sending queries of a FQDN on the same *Home* node, without modifying the core network infrastructure of ISPs, we consider Pastry [20] based architectures. In Pastry based architecture, queries are identified according to the requested FQDN, and each node resolves queries it *Home*s and routes other queries to their *Home* node.
Pastry is widely known by the community, but other DHT protocols may be used - like chord, Tapestry... The way we use Pastry differs from what it was originally designed for. First, the Pastry nodes constitute the platform and belong to the same administrator, they are located in the same data center, on the same LAN and every node knows all the other nodes - the platform is not expected to be larger than a few hundred nodes. We do not assert that the node ID is derived from the data by a simple hash function (SHA1), but we may apply a specific distribution known by all nodes. Finally, we do not consider the Pastry routing discovery algorithm. On the other hand, we take advantage of Pastry's auto-configuration mechanisms, robustness to DoS attacks [14], [17], [18] - as such it may balance the sensitivity to DoS attacks introduced by DNSSEC with heavier resolutions. We also take advantage of the cache sharing mechanisms for enhancing Pastry based platforms. Note that how FQDNs are associated to their *Home* may require the autoconfiguration mechanism to be reconsidered. Furthermore, robustness to DoS requires architectures that cache somehow the responses.

The pro-active Pastry architecture is described in [7], [8] and consists of caching in all pastry nodes the most γ popular data. This architecture is especially efficient when requested data follows a Zipf distribution. Figure 2b illustrates the principle of Pastry Active Caching (*Pastry-AC*). Figure 2a shows the popularity distribution of the FQDN whose Zipf-like shape encourage to test the pro-active caching architecture.

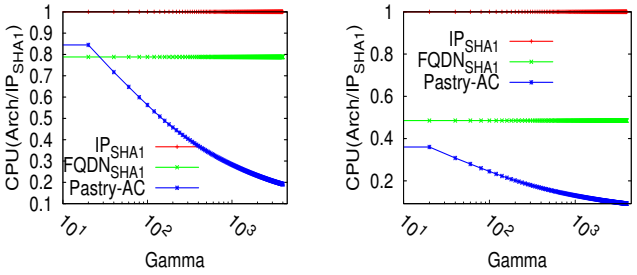
To compute figure IV, we modelize the *Pastry-AC* architecture and compute it our the measured popularity distribution of our 10 minute capture. Figure IV shows that caching very FQDNs considerably enhances the performances. For



(a) FQDN Popularity Distribution (b) Pastry Active Caching: n_j

Fig. 2. DNS traffic / Pastry Active Caching

DNSSEC in figure 3b, caching as little as $\gamma = 100$ makes *Pastry-AC* twice as much efficient as $FQDN_{SHA1}$. Interestingly *Pastry-AC* also provides advanatge for DNS and caching $\gamma = 100$ makes *Pastry-AC* makes *Pastry-AC* as efficient as $FQDN_{SHA1}$. Given the small amount of FQDNs to be cached, this number can be increased to $\gamma = 200$ and thus one can reasonably consider that for DNSSEC *Pastry-AC* can reasonnably be considered as 3.5 more efficient than $IP_{SHA1/XOR}$.



(a) γ_{Nbr} (DNS) (b) γ_{Nbr} (DNSSEC)

Fig. 3. Evaluation of the impact of γ on CPU

V. Conclusion

DNSSEC migration of large resolving platforms is expected to increase their size by 5 if one keeps on using the current DNS Resolving platform architecture. This paper first evaluates different load balancing techniques and shows that load balancing according to the FQDN rather than according to IP addresses reduces the number of nodes by 30%.

Because FQDN load balancers significantly impact the ISP core network infrastructure and still provide a single point of failure, we considered Pastry based architectures and their associated cache sharing optimizations. With the Zipf distribution of FQDNs, we showed that *Pastry-AC* with Active Cache sharing mechanisms can be at least 3.5 times more efficient than traditional IP architectures.

We also have to consider such results regarding the approximations we performed. We considered a constant TTL. Then, we did not consider the impact of the size of the cache on cache lookup or cache insertion. At last, we also considered that IP addresses and FQDN were independent.

References

[1] D. Kaminsky, "It's The End Of The Cache As We Know It, or 64K Should Be Good Enough For Anyone," URL: http://www.doxpara.com/DMK_BO2K8.ppt, IOActive, july 2008.

[2] K. J. Higgins, "Kaminsky Calls For DNSSEC Adoption," URL: http://www.darkreading.com/security/vulnerabilities/showArticle.jhtml?articleID=214501924, DarkReading, february 2009.

[3] D. Migault, C. Girard, and M. Laurent, "A performance view on dnssec migration," in *CNSM 2010*, oct 2010, pp. 469–474.

[4] "Radaware Alteon Application Switch 5412 Case study," URL: http://www.radwarealteon.com/wp-content/uploads/Documents/CaseStudies/NAS/Alteon_Case_Study.pdf, 2010.

[5] S. Francfort, D. Migault, and S. Sénécal, "A bi-objective mixed integer linear program for load balancing dns(sec) requests," in *International Journal of Critical Infrastructure Protection*. Elsevier, 2012.

[6] F5-Networks, "F5, Networks: BIG-IP Global Traffic Manager: Implementation," march 2010.

[7] V. Ramasubramanian and E. G. Sirer, "Beehive: O(1)lookup performance for power-law query distributions in peer-to-peer overlays," in *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*. Berkeley, CA, USA: USENIX Association, 2004, pp. 8–8.

[8] ——, "Proactive Caching for Better than Single-Hop Lookup Performance," URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=?doi=10.1.1.2.8918, 2007.

[9] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: evidence and implications ," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, march 1999, pp. 126 –134 vol.1.

[10] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS performance and the effectiveness of caching," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, ser. IMW '01. New York, NY, USA: ACM, 2001, pp. 153–167.

[11] J. Jung, A. W. Berger, and H. Balakrishnan, "Modeling TTL-based Internet Caches," in *IEEE Infocom 2003*, San Francisco, CA, april 2003.

[12] K. Park, V. S. Pai, L. Peterson, and Z. Wang, "CoDNS: Masking DNS delays via Cooperative Lookups," in *Technical Report TR-690-04*. Princeton University Computer Science, 2004.

[13] ——, "CoDNS: improving DNS performance and reliability via cooperative lookups," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*. Berkeley, CA, USA: USENIX Association, 2004, pp. 14–14.

[14] R. Cox, A. Muthitacharoen, and R. Morris, "Serving DNS Using a Peer-to-Peer Lookup Service," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, ser. IPTPS '01. London, UK: Springer-Verlag, 2002, pp. 155–165.

[15] I. Stoica, R. Morris, D. Karger, F. M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, vol. 31. New York, NY, USA: ACM Press, october 2001, pp. 149–160.

[16] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, ser. SOSP '01. New York, NY, USA: ACM, 2001, pp. 202–215.

[17] D. Massey, "A Comparative Study of the DNS Design with DHT-Based Alternatives," in *In the Proceedings of IEEE INFOCOM'06*, 2006.

[18] J. Risson, A. Harwood, and T. Moors, "Topology dissemination for reliable one-hop distributed hash tables," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, pp. 680–694, 2009.

[19] Q. Xu, D. Migault, S. Sénécal, and S. Francfort, "K-means and adaptive k-means algorithms for clustering dns traffic," in *Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools*, ser. VALUETOOLS '11. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), may 2011, pp. 281–290. [Online]. Available: http://dl.acm.org/citation.cfm?id=2151688.2151720

[20] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *Lecture Notes in Computer Science*, vol. 2218, p. 329, 2001.