ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 3, March 2017

Analyzing Internet DNS (SEC) Traffic with "R" For Resolving Platform Optimization

[1]J Uma Mahesh, [2]Harekrishna Allu ,[3]N Chandrakanth
[1][2]Assistant Professor [3] Professor
Department of CSE, Geethanjali College of Engineering and Technology

Abstract— This paper proposes to use data mining methods implemented via R in order to analyze the Domain Name System (DNS) traffic and to develop innovative techniques for balancing the DNS traffic according to Fully Qualified Domain Names (FQDN) rather than according to the Internet Protocol (IP) addresses. With DNS traffic doubling every year and the deployment of its secure extension DNSSEC, DNS resolving platforms require more and more CPU and memory resources. After characterizing the DNS(SEC) traffic thanks to reduction in dimension and clustering methods implemented with R functions and packages, we propose techniques to balance the DNS traffic among the DNS platform servers based on the FQDN. Several methods are considered to build the FQDN-based routing table: K- means clustering algorithm, mixed integer linear programming, and a heuristic scheme. These load balancing approaches are run, and evaluated with R on real DNS traffic data extracted from an operational network of an Internet Service Provider. They result in reducing the platform CPU resources by 30% with a difference of less than 2% CPU between the servers of a platform.

***Index Terms*— Telecommunications; Internet; DNS; DNSSEC; Feature selection; Dimension reduction; Clustering; Load balancing; K-means.**

I. INTRODUCTION

Domain Name System (DNS) (Mockapetris, 1987a,b) is the computer protocol that facilitates Internet communication using hostnames by matching an Internet Protocol (IP) address and a Fully Qualified Domain Name (FQDN), e.g., ―www.google.com.‖ DNS servers, which host the IP addresses of the queried web sites—that is to say the DNS responses—are called Authoritative Servers. Because Authoritative Servers would not be able to support all end users' queries, the DNS architecture introduces Resolving Servers that cache the responses during Time to Live (TTL) seconds. Internet Service Providers (ISPs) manage such servers for their end users. Thanks to the caching mechanism, Resolving Servers do not need to ask Authoritative Servers if the response is still in their cache. This provides faster responses to the end user and reduces the traffic load on the DNS Authoritative Servers.

For multiple reasons, ISPs consider operating DNSSEC, the security extension of DNS defined in the standards (Arends et al., 2005a,b,c; Sawyer, 2005). With DNSSEC, a DNS response is signed so that its authenticity (generation by a legitimate Authoritative Server) and its integrity (nonmodification of response) can be checked. With DNSSEC, resolutions require multiple signature checks so that responses are around seven times longer than traditional DNS responses. Migault (2010), Migault et al. (2010), and Griffiths (2009) show that DNSSEC resolution platforms require up to five times more servers than DNS resolution platforms. Migault et al. (2010) measures that a DNSSEC resolution involves three signature checks and costs up to 4.25 times more than a regular DNS resolution. With the DNS traffic doubling every year and the deployment of its secure extension DNSSEC, DNS resolving platforms require more and more resources.

The operational problem faced is to reduce the resources needed by a resolving platform. The resolving platform consists of several DNS resolving servers behind a load balancer device. The load balancer splits the incoming traffic to distribute queries on resolving servers. The classical way of load balancing is performed by assigning a pool of clients to be served to each server.
One way to reduce the load on a server is to lower the number of resolutions. To reduce the number of resolutions, Migaultand Laurent (2011) and Francfort et al. (2011) evaluate the advantage of splitting the DNS traffic according to the queried FQDN rather than according to the IP addresses. This increases the efficiency provided by caching mechanisms, reduces the number of signatures to be checked, and can result in a 1.32 times more efficient architecture.

To design this new load balancing mechanism, we first need to characterize the DNS traffic and to evaluate how the DNSSEC traffic looks like. We perform data

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 3, March 2017

extraction from raw network captures taken from a DNS resolving platform. The main challenge here is to define the variables, which are taken and computed for each FQDN. The goal is to define a routing table mapping each frequently requested FQDN to a server of the resolving platform.

II. DATA EXTRACTION FROM PCAP TO CSV FILE

To conduct this study, we first gather pieces of DNS data. They consist of real outbound and inbound DNS traffic of the platform stored in PCAP files. Then, for each FQDN found in a traffic sample, we compute a series of variables. Given the application considered, these variables are related to the FQDN‘s resolution cost.

Network costs: servers occupation times associated to a FQDN (time between a query and its response) and different rates:

- mean open context times observed for resolvers: Mean Internet Resolution Time (MIRT) and Mean Platform Resolution Time (MPRT)
- end user and platform query rates (euQR and reQR)
- end user and platform bit rates (euBR and reBR) Computation costs: signature checks related variables:
- number of signature checks (SigCheck)
- cache hit rate (CHR)
- Memory costs: cache length and cache update related variables:
- mean TTL observed (MTTL)
- query and response length (Qlen and Rlen)
- response time for cached response

These variables are exported into a CSV file. CSV is a standard format that can be read from many softwares and languages, including R. To generate this CSV file, we use a homemade python script. This file is composed of lines terminated with the UNIX-compliant end of line character (—\n‖), each line containing the variables corresponding to a FQDN into fields. The separator that separates fields is the classical space and fields are not enclosed between quotation marks. The first field is the FQDN, i.e., the label of the vector corresponding to the FQDN. Variables labels are not included in the CSV file to ease some common operations like split or concatenation of several CSV input files. The first arrow in Figure 1 represents this step.

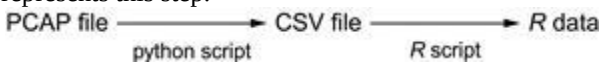

FIGURE 1 Extraction and importation from PCAP files to R.

Our dataset is now stored in a CSV file that consists of vectors corresponding to FQDN and composed of cost- related measures.

III. DATA IMPORTATION FROM CSV FILE TO R

Once the dataset is extracted from PCAP files to CSV files, we import these files into R. To do so, we use the code in the Listing 1. This step is represented by the last arrow in Figure 1. As described in Section 1.2, the CSV file does not contain dimension labels. In line 3 of Listing 1, we construct a vector containing all labels in the correct order. These labels will be used.

Listing 1
R Code Used to Load Dataset from CSVFile

```
filename = -inputfile.CSV‖ # input filename
clab<- c(-euQR‖, -reQR‖, —tR‖, -euBR‖, -reBR‖,
-tBR‖,
-cQRT‖, -reOCC‖, -euOCC‖, -tOCC‖, —CHR‖,
-eQR‖,
-eCPU‖, -ePRT‖, -succRatio‖, -failRatio‖,
-cltQNbr‖,
-pltQNbr‖, -Qlen‖, -Rlen‖, —Sigcheck‖, -MIRT‖,
-SDIRT‖,
-MPRT‖, -SDPRT‖, -MTTL‖, -SDTTL‖)
mat_ent<- read.table (filename, row.names=1,
col.names=clab)
mat_ent<- subset (mat_ent, cQRT> 0) # python script
return − 1 if no request is present and cQRT is used to plot
several variables
mat <- subset (mat_ent, MTTL > 0) # remove non valid TTL
```

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 3, March 2017

Later to ease dimension selection. The R-function used to import data is read.table(), in line 5. This function returns a matrix stored in a data.frame object whose dimensions are labeled thanks to therow.names and col.names arguments. We set the name of the input file at the beginning of our code (line 1) to ease the readability and the further modification of the input file name. Note that the way our CSV file is defined allows us to keep default values for most of the read.table() function‘s parameters. The last lines of Listing 1 (lines 7 and 8) are used to remove lines (i.e., FQDN), which present non acceptable values for the variables, from our data. This is a step to delete FQDN whose variables are not coherent or not in the expected intervals. We now have a data.frame containing the input dataset for further R processing.

IV. DIMENSION REDUCTION VIA PCA

The dataset consists of several thousands of 27-dimensional vectors, each vector corresponding to a FQDN. For a better understanding, we aim at reducing this dataset volume by shrinking the number of its dimensions, i.e., the number of FQDN characteristics. To perform this dimension reduction, we use principal component analysis (PCA; cf. Cox and Cox, 2001), for instance. PCA is an efficient way to reduce the number of noninformative dimensions and to eliminate correlated variables. The PCA algorithm is implemented in R through the function prcomp(). The code used to perform PCA is presented in Listing 2.

Listing 2
R Code Used for PCA

```
r = 0.9  # threshold for PCA
output_file = paste (format (Sys.time(), ―%F-%T‖), ―-
Rout.txt‖, sep=―‖) # file where to
print tmp_file = ―/tmp/foo‖ #tmp
file sink(output_file)
clabf<- c(―euBR‖, ―reBR‖, ―QNbr‖, ―pltQNbr‖, ―CHR‖,
―cQRT‖, ―MIRT‖, ―MPRT‖,
―MTTL‖) mat <- subset (mat,
select=clabf)
pca<- prcomp(mat, scale=TRUE, center=TRUE)
mag <- sum(pca$sdev * pca$sdev)  # total magnitude
pca<- prcomp(mat, tol = (1 − r)*mag/(pca$sdev
[1] * pca$sdev [1]), scale=TRUE, center=TRUE)
write.table(pca$x, file=tmp_file)
d<-read.table(tmp_file, header=TRUE, row.names=1)
write.table(pca$rotation, file=tmp_file)
rot<-read.table(tmp_file, header=TRUE,
row.names=1) print(pca$rotation) # new vectors
sink()
```

To ease the exploitation of the PCA‘s results, we dump the output stream into a file which can be read thanks to any text editor. To do so, we first construct the name of this file. We want to keep and distinguish results from this script run at different times. The filename (line 2 of Listing 2) contains the date when the script is run (Sys.date()) in a friendly format (format()). This timestamp is concatenated with another string (“-Rout.txt”) thanks to the function paste. We change the separator of this function to the empty string (sep=“ ”) to avoid space in the filename.

We also use a temporary file to write and read some data. This is a trick to reformat a matrix object into a data.frame object which can be avoided using as.data.frame(). Moreover as.data.frame() takes less time as it does not require hard disk access. To open an output flow, we use the function sink() (line 4 ofListing 2) with the filename constructed line 2. This redirects all the output into the file. Note that the file should be closed (line 18 of Listing 2). As a preparation step, we also store a subset of dimension labels in a vector (line 5) to be used later to select a subset of initial data thanks to the subsetfunction (line 7). This subset concerns only nine variables and ignores others which are linear combinations of the first ones.

In the PCA, we define a threshold to decide which components are kept. We aim at keeping a percentage of the total magnitude. We used prcomp() the first time (line 9 of Listing 2) to get the whole magnitude, i.e., the whole variance, and to compute the variance kept. We recompute a PCA using this variance value (line 11). The result of this step is the rotation matrix and the vectors in the new basis. These results are stored in variables (line 13 of Listing 2) and printed into the output file (line 16) (Figure 2).

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 3, March 2017

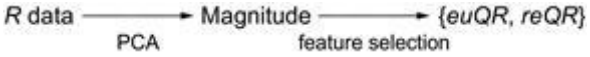

FIGURE 2 Reduction dimension via PCA.

Thanks to the rotation matrix and the screegraph (plot of variance explained by each principal component represented in Figure 3), we can see that:

- The two first principal components hold a significative part of magnitude (35% in our application case)
- The two first principal components are mainly due to euQR and reQR

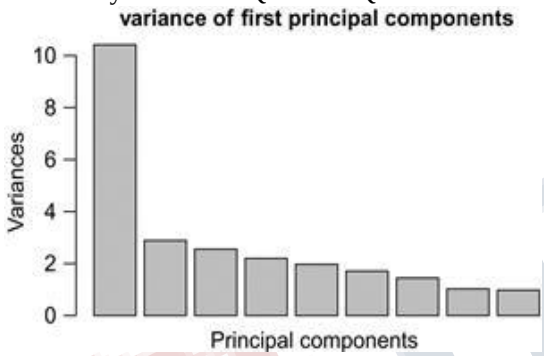

FIGURE 3 Screen graph of PCA on initial variables.

This result could have been obtained by analyzing the variance for each variable individually. euQRand reQR are the most discriminative variables from the viewpoint of second-order statistical information (variance).

V. INITIAL DATA EXPLORATION VIA GRAPHS

To be more familiar with the data considered in this problem, also to learn how they look, and to define which process to apply, we perform an exploration phase. We conduct this initial exploration through graphs. All graphs performed with R are drawn into a postscript file to be edited with external tools if needed. There are multiple types of graphs depending on what we plot with R and what parameters we provide to the plot() function. To draw points, we provide the list of coordinates (list of abscissas and list of ordinates) to the plot()function. For a matrix or a data.frame, the plot() function performs a scatterplot. This consists of a series of graphs, each being the representation of data in a two-dimensional space. All possible couples are represented. Such a graph can be seen in Figure 4.

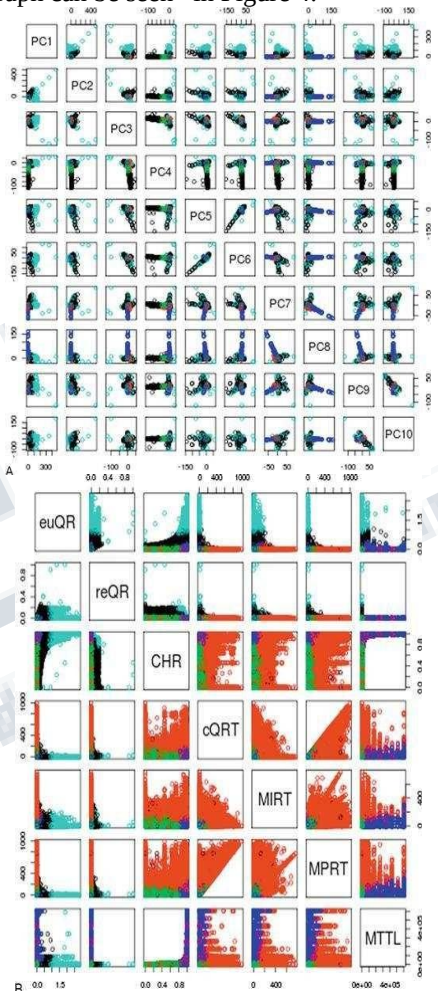

FIGURE 4: Multidimensional representation of FQDN. (a) Principal components. (b) Subset of initial parameters.

The function density() returns a kernel density estimate which is drawn with the plot() function. This graph is useful to know if the distribution is multimodal (see Figure 5).

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 3, March 2017

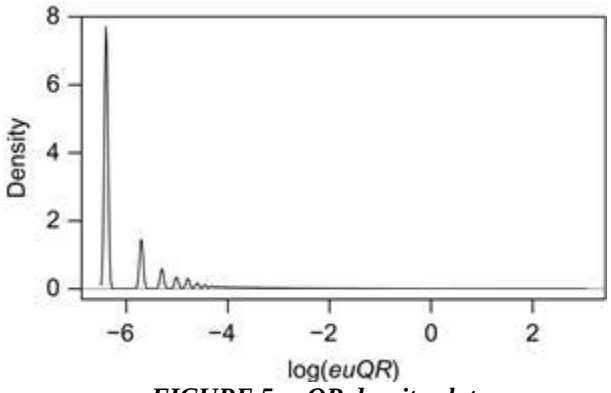

FIGURE 5 euQR density plot.

Boxplots highlight median, quartiles, minimum, maximum, and outliers. When the input of theboxplot() function is a data.frame, it traces a boxplot for every dimension. This simple drawing allows us to see immediately if a dimension seems discriminant and highlights outliers and imbalances in the distribution. This representation also helps to visualize the differences between variables.
Thanks to these simple graphs, we can define further processes to apply to the dataset to get more balanced features.

VI. VARIABLES SCALING AND SAMPLES SELECTION

As seen in Section 5, all the variables are not equivalently informative to discriminate the FQDN, see, for instance, Figure 4b. Moreover, the distribution of the queries and the responses rates highlighted in Section 5 suggests that these variables should be processed using a log function. Indeed, the range and the distribution of values for these variables do not give an informative representation. In this case, a standard linear representation is not very relevant. Instead, we choose to apply a logarithmic transformation to grasp more precisely the value amplitudes for the variables of interest. We also decide to remove the less requested FQDN (euQR less than a threshold) because many FQDN are requested only a couple of times during the timeslot used for the traffic capture. The code used to perform this processing is presented in Listing 4. We add to the original data (stored inmat) three variables (cf. lines 1-3 of Listing 4).

Listing 3

```
Generation of Several Graphs intoPostscript Files
postscript (—pca_magnitude.ps‖) plot(pca)
dev.off ()
postscript (—boxplot.ps‖) boxplot (mat)
dev.off()
postscript (—scatter.ps‖) plot(mat)
dev.off()
postscript (—euQR_density.ps‖)
plot (density (log(mat$euQR) ), xlab=— log (euQR), main=—
‖) dev.off() Listing 4
Application of log() on Initial Dataset, Sample Selection, and Feature Selection
mat$logeuQR<− log (mat$euQR) mat$logreQR<− log (mat$reQR) mat$logSigcheck<− log (mat$Sigcheck) mat <− subset (mat, euQR> threshold)
m_mat<− subset (mat, select=c (—reQR‖, —euQR‖) )
```

Once PCA has been applied, we select the most informative variables for the problem considered.euQR is the variable with the greatest variance. The operation consisting in applying the log() function and removing the less requested FQDN (lines 5 and 6 of Listing 4) can be considered as preprocessing. To visualize the effects of this preprocessing, we use histograms (Figure 6). As the kernel smoother used by density(), histogram is a density estimator and allows us to visualize the distribution (Figure 7). us from finding objects in the original space that are not in our dataset. It also enables the precomputation of all the samples interdistances (i.e., the use of a dissimilarity matrix). The K-means algorithm is implemented in R through the function kmeans(). This function is part of thestats package (R Development Core Team, 2010). This function returns a kmeans object consisting of clusters and some cluster characteristics. Also, the K-medoïds algorithm is implemented through thepam() function. pam stands for Partition Around Medoïds. This function is provided by the packagecluster (Maechler et al., 2005) loaded in line 1 in Listing 5. The pam() function returns a cluster object.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 3, March 2017

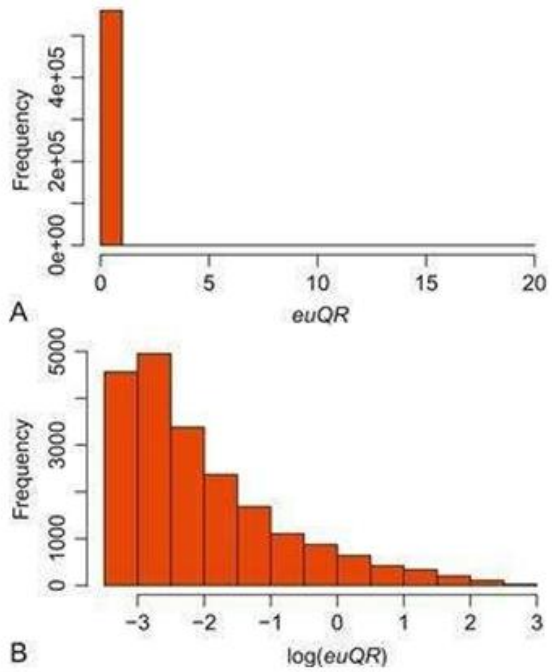

FIGURE6 Preprocessingthe variable euQR. (a) euQR (q s− 1) without any transformation. (b) log(euQR) without less requested FQDN.

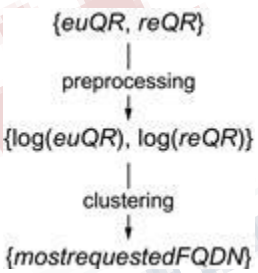

FIGURE 7 Preprocessing and clustering after feature selection.

Listing 5
Clustering and Silhouette Visualization

```
library(—cluster)
# silhouette width for pam and kmeans swlqr<− numeric(25) kswlqr<− numeric(25) sink(file=output_file, split=TRUE) for (k in c(2:3) ) {
# kmean log qr
km <− kmeans(clqrmat, center=k, iter.max=1000) kswlqr[k] <− summary(silhouette (km$cluster, daisy(clqrmat) ) ) $avg.width
png(paste(—k0‖, k, —qr_log_kmean.png‖, sep=—‖) ) par(cex=2); plot(qrmat, col=km$cluster * 5, log=—xy‖, pch=km$ cluster)
dev.off()
print(paste(— - log qrkmean - k =‖, k) ) mysummarykmean(km)
# kmed log qr
km <− pam(clqrmat, k=k)
swlqr[k] <− km $ silinfo $ avg.width png(paste(—k0‖, k, —qr_log_kmed.png‖, sep=—‖) ) par(cex=2); plot(qrmat, col=km$ clustering * 5, log=—xy‖,
pch=km$ cluster) dev.off()
print(paste(— - log qrkmed - k =‖, k) ) print(km$clusinfo)
```

VII. CLUSTERING FOR SEGMENTING THE FQDN

The goal pursued is to separate FQDN into different groups depending on their costs. The initial idea we As suggested in Section 6, we cluster the data using investigate is to define for each FQDN a set of cost-related variables (Section 6) and to cluster FQDN using an unsupervised machine learning technique. We aim at clustering the data in groups of FQDN having the same cost origins (e.g., frequently requested, long response, low TTL).

We use simple clustering algorithms: K-means and K-medoïds, for instance (cf. Hastie et al., 2008;Kogan, 2007). The K-means is a clustering algorithm grouping similar pieces of data together. A group is characterized by its centroïd which is a vector minimizing the distances to all other elements of the group. The K-medoïds algorithm uses medoïds instead of centroïds. The difference between centroïds and medoïds is that medoïds are necessarily points belonging to the initial dataset. This characteristic prevents a subset of initial variables (euQR and reQR) and another subset of preprocessed variables (log(euQR) and log(reQR)). In practice, the K-means and K-medoïds algorithms applied to the dataset exhibits a convergence in less than 15 iterations. As a result, the data are well segmented into groups corresponding to the FQDN which are either rarely requested or frequently requested among the traffic (cf. Xu

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 3, March 2017

et al., 2011). We also observe that the K-means and the K-medoïds schemes converge to similar clustering results. It can be explained by the samples distribution and shape of the data, in which the centroïds are located quickly quite close to the medoïds data points. To determine the relevant number of clusters, we used the silhouette as defined in Rousseeuw (1987). The silhouette is defined for each sample and takes values between − 1 and 1.

- it is close to 1 when the sample is near the center of the cluster it belongs to.
- it is almost null if the sample is located near the frontier between its cluster and the nearest cluster.
- it is negative if the sample is in a cluster it should not belong to.

For each FQDNi in cluster Ci, we measure ai the average distance between FQDNi and other FQDN ofCi. ai measures the average dissimilarity of FQDNi with Ci. Then, we measure bi the minimum average distance between FQDNi and other FQDN in clusters (Cj)j≠ i. bi measures similarity with other clusters. The silhouette for FQDNi is given by:

$$s_i = \frac{b_i - a_i}{\max\{a_i, b_i\}}$$

By construction of the K-means and the K-medoïds algorithms, the silhouette cannot be negative. We run the clustering algorithms for several numbers of clusters (for loop from lines 9 to 27 in Listing 5). At each iteration, we compute the average silhouette and store the result in a vector (created lines 4 and 5). For human readability, we draw the average silhouette thanks to the code presented in Listing 6. To monitor the evolution of the for loop, we Listing 6

R Code Used to Plot Silhouette

```
# plot barplot of sil value for k in c(2:15) for aabb<− mat. or .vec(2,15)
aabb[1, 1:15] <− kswlqr[1:15] aabb[2, 1:15] <− swlqr[1:15] par(cex=2)
barplot(aabb[,2:15], beside=TRUE, col=c(―dark blue‖, ―pink‖), names.arg=c(2:15), xlab=―cluster number‖, ylab=―average silhouette width‖, legend=c(―kmean‖, ―kmedoid‖) )
```

decide to split the output flow. The argument of the sink() function (line 7) makes two identical copies of the output flow:

- one flow for the standard output to monitor the evolution of the R script.
- one flow written in the file whose name is stored in the variable output_file.

As the objects returned by kmeans() and pam() are not the same, the silhouette is not computed the same way. For the cluster object returned by the pam() function, we immediately access the silhouette information (line 21). For the kmeans object returned by kmeans(), we compute the silhouette thanks to the silhouette() function included in the R package cluster (Maechler et al., 2005). We provide to thesilhouette function clusters as returned by kmeans() and dissimilarity between samples computed bydaisy(). daisy() is also part of the cluster package (Maechler et al., 2005). We use the summary() function because the summary. silhouette object returned is easier to manipulate than the silhouette object returned by the silhouette() function. This is illustrated in line 12 of Listing 5.

We now handle silhouette values for multiple numbers of clusters (k values) and for the two clustering algorithms (K-means and K-medoïds). To visually compare the results, we use barplot(). The code used is written in Listing 6. First, we cast all data into a two-dimensional array. This array is declared and filled (lines 3-5 fromListing 6) with data from Listing 5. To enhance readability, we increase label size thanks to thepar() function (line 6). This function controls layout parameters for graphs. The cex parameter controls the size of text and symbols. The colors used (dark blue and pink) are chosen to be quite different if the graph is printed in black and white. The results are presented in Figure 8. They show that the highest silhouette values for both clustering algorithms are obtained for 5, 3, and 2 clusters. This gives reliable estimates of the number of clusters fixed apriori to run the clustering algorithms. We perform an analysis of the DNS traffic through feature selection and clustering in

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 3, March 2017

Section 4 and above. Now, we devote the three following sections to the construction of a routing table for the identified heavily requested FQDN.

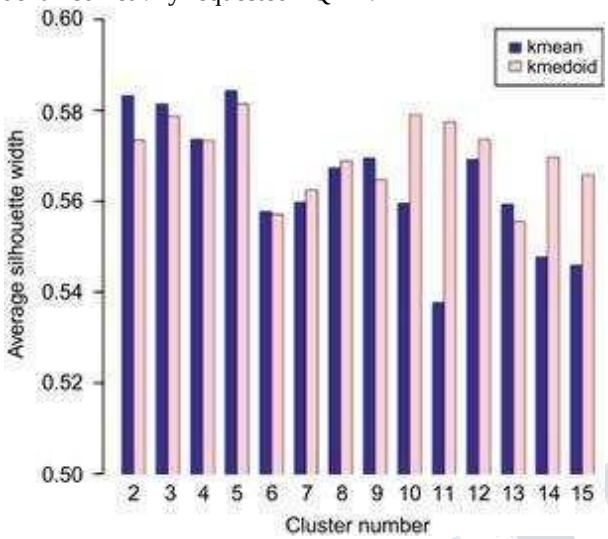

FIGURE 8 Average silhouette versus number of clusters.

VIII BUILDING ROUTING TABLE THANKS TO CLUSTERING

As explained in Section 1, our goal is to build a routing table for the most requested FQDN to balance the load of the incoming DNS traffic in our resolution platform. The routing table is a function mapping a FQDN to a server of the platform, the platform being a set of resolution servers. This mapping is composed of explicit entries mapping FQDN to servers. For FQDN, which are not frequently requested, we compute the mapping on the fly based on a hashing function. We focus on the most requested FQDN for building the explicit mapping. Our first idea is that clustering outputs homogeneous groups of FQDN, each one having a different main source of cost. To balance the resources used between the servers of the platform, the idea is to distribute each group of FQDN homogeneously between servers. Doing this should dispatch the consumption of each kind of resource (network resources, memory, CPU, etc.) equally on each server. Unfortunately, as shown in Section 4, two variables (euQR and reQR) are more discriminative than the other because of their variance. To build the routing table, we proceed cluster by cluster. For each cluster, we distribute FQDN in a round-robin fashion. The algorithm is detailed in algorithm 1.

This algorithm outputs a routing table for the frequently requested FQDN, which maps the FQDN to different resolving servers. This table is not used directly after its generation but will be considered inSection 11 to be compared with routing tables built thanks to other approaches.

Programming (MILP) method (cf. Schrijver, 1998) for instance:

I: Set of FQDN requested by the end users
J: Set of servers composing the DNS Resolving Platform
qi, i ∈ I: Queries number associated with FQDN i
ri, i ∈ I: Number of resolutions associated with FQDN i X = xi,j, (i, j) ∈ I × J: Matrix binding FQDN i to server j Qj, j ∈ J: Number of queries supported by server j
Rj, j ∈ J: Number of resolutions supported by server j
We have immediately:

Algorithm 1 Building routing table based on clustering

Require: cluster_number // number of clusters for the clustering algorithms
Require: server_number // number of servers in our platform server ← 1 // used to indicate to which server current FQDN will be mapped
for k = 0 to cluster_number do
for fqdn ∈ k // enumerate FQDN belonging to cluster k do maps FQDN fqdn to
server server mod server_number // add an explicit entry for the mapping
server ← server + 1
end for end for

IX BUILDING ROUTING TA BLE THA N KS TO M I X E D INTEGER LINEAR PROGRAMMING

Another approach to building an efficient mapping between the FQDN and the servers of the resolving platform is to use linear programming. This idea is driven by the fact that we face an optimization problem. We used GLPK (Theussl and Hornik, 2010) to solve this problem. AlthoughGLPK can be used as an R package, we use it as a standalone program.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 3, March 2017

Operational teams evaluate the efficiency of different load balancing techniques by comparing the CPU load of each server. However, providing an estimation of the CPU load for a server relies on experimental measurements, and as (Migault et al., 2010) mentioned, measured values for the CPU load depend on the hardware, the DNS server implementation, the nature of the traffic, etc. Since we do not want to depend on these factors, we evaluate the difference by considering the number of queries and resolutions performed by each server of the platform. Such evaluation requires defining specific notations we will use in the later in this chapter. Furthermore, these notations are also used to build a routing table with a mixed integer linear

We use this method to build a routing table for the most requested FQDN milp-200 as we consider 200 FQDN. This number is the result of an operational evaluation. It is a compromise between the minimization of the computation time and the minimization of the number of FQDN that are not balanced thanks to the routing table. For each FQDN, the number of resolutions is computed thanks to the number of queries and the mean TTL value observed for the FQDN. Because we consider the popular FQDN, we assume that a resolution occurs every TTL seconds.

Although this method makes it possible to build a routing table thanks to a technically sound scientific approach, in practice it happens to be heavy to implement because of the computational burden which limits its applicability. Also, this method needs to evaluate a priori the number of FQDN to be processed, which relies only on an empirical estimation.

The MILP method is based on solving a system of equations. We define a given set I of FQDN (line 1 ofListing 7). For a given distribution of these FQDN on the servers (xi,j)(i,j)∈I× J, we compute the number of queries and resolutions supported by each server (Qj, Rj)j∈ J. The distribution we seek minimizes the differences between the servers of the platform in terms of (Qj, Rj)j∈ J: ΔQ and ΔR.

Mixed Integer Linear Pro gram Used toBuild a Routing Table

```
set I;      /* set of fqdn */ set J;      /* set of servers */
paramk;  /* k parameter */
paramc{i in I, j in 1..2}; /* costs[r, q] */ varS{j in J};  /*
sum of requests */ varT{j in J};  /* sum of resolutions */
varx{i in I, j in J} binary; /* 1 if Fi affected to Sj */
vardeltar;
vardeltaq; var max;
minimize cost : max; /* objectives */
s.t.slack{(j1, j2) in (J cross J)} : k * S[j1] + (1 - k) * 10000 *
T[j2] <= max;
s.t.aff{i in I} : sum{j in J} (x[i, j]) >= 1; /* one fqdn affected
to at least 1 server */
s.t.q{j in J} : sum{i in I} (x[i, j] * c[i, 1]) = S[j] ;
s.t.r{j in J} : sum{i in I} (x[i, j] * c[i, 2]) = T[j] ;
solve;
end;
```

Trying all possible combinations for such a distribution is not feasible, so we formulate our problem as a MILP and use a solver (GLPK in this case, Theussl and Hornik, 2010) to find a proper distribution. Although an R application programming interface for GLPK exists (Theussl and Hornik, 2010), we decide not to use it. Writing the problem using the GNU Mathematical Programming Language, the native language for GLPK, is easier once the problem is modeled.

The challenge for the solver is to find a distribution that is close to the optimal distribution, even though we do not know the optimal solution. Considering 200 FQDN is a compromise between the total number of FQDN to process and the resources needed for the computation as shown in (Francfort et al., 2011). The integer linear optimization problem with two objectives consists in minimizing ΔQand ΔR defined as follows:

$$\Delta Q = \max_{j \in J} Q_j - \min_{j \in J} Q_j$$

$$\Delta R = \max_{j \in J} R_j - \min_{j \in J} R_j$$

To ease the resolution by the solver, we reduce the number of objectives by defining a FQDN cost:

λ is a weighting parameter which determines the parts of q and r in the definition of the cost. In that sense, the important parameter is not λ itself but the ratio.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 3, March 2017

$$\frac{\lambda}{1-\lambda} \left(or \ \frac{1-\lambda}{\lambda}\right)$$

Note that λ is introduced here only for resolving purpose, and has a priori no physical meaning. The problem is then rewritten as:

$$\text{minimize} : \max_{j \in J} C_j$$
$$\text{with} : \forall j \in J, C_j = \sum_{i \in I} x_{i,j} c_i$$

Cj represents the cost supported by the server j. This objective function ensures the minimization of the cost supported by each server, which leads to the minimization of the difference of costs between the servers of the platform. The cost that is not supported by the most loaded server is reported on other servers, increasing the cost supported by the less loaded server, which thus reduces the difference of costs supported by the servers.

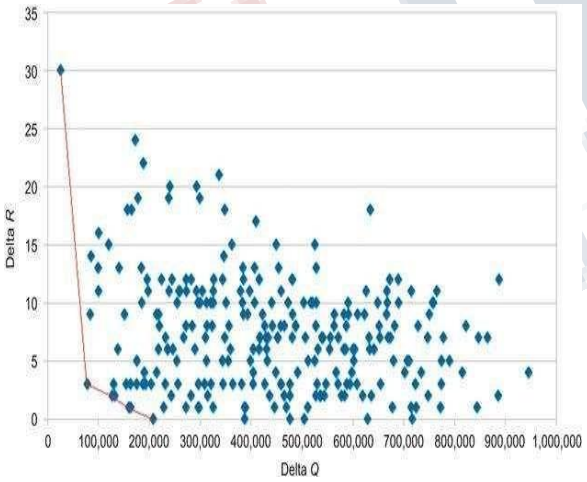

FIGURE 9 Bi-criteria MILP results.

Building Routing Table via a Heuristic

The method described in Section 9 gives us promising results, but can only consider a limited number of FQDN. We thus evaluate another approach based on a heuristic.

The goal of this algorithm is similar to milp-200: minimizing jointly ΔQ and ΔR. However, the way we build the routing table provides less accurate results as milp-200. As a result, we need to consider a much larger set—namely, 18 times larger—of FQDN to build a routing table which balances properly the load among the servers. Even though the routing table is roughly 18 times larger, it takes less than 0.5 s to build it. Compared to 1000 s with milp-200, this method may present an operational advantage over milp-200. The algorithm starts with I, the set of the most requested FQDN. From the current set of FQDN, it takes the costliest FQDN, assigns it to the less charged server (i.e., a server jmin verifying) and removes it from the FQDN set. This step is performed until the set of FQDN is empty.

$$C_{j_{min}} = \min_{j \in J} C_j$$

To compare this method with milp-200, we compute ΔQ and ΔR for different values of λ. We first choose a set of 200 FQDN to be compared with the results from Section 9. Then, by construction, an upper bound of the difference between the cost on different servers is mini∈ I ci, ci being the cost of the less costly FQDN. Thus, we choose I, the set of FQDN, such that the last element is associated with a cost that is roughly the difference of costs generated by milp-200. This leads to consider 1580 FQDN. We denote these algorithms as stacking-200 and stacking-1580. Note that 200 FQDN represent 16% of the number of queries and that 1580 FQDN represent 46% of the number of queries. Figure 10shows that stacking-200 presents a lower front compared to stacking-1580. However, ΔQ and ΔR are computed according to the set of FQDN I. This set is definitely not the same in stacking-200 and instacking-1580, which makes the comparison between stacking-200 and stacking- 1580 difficult according to Figure 10.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 3, March 2017

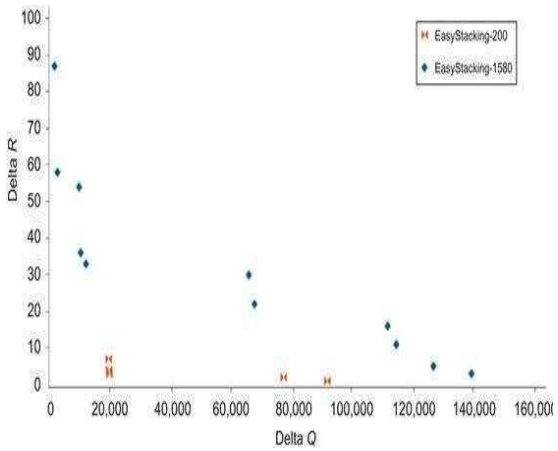

FIGURE 10 Pareto front with easy stacking.

One must keep in mind that this comparison takes into account the FQDN used for building the routing table. This is motivated by the fact that we evaluate the routing table and not the imbalance owing to the less requested FQDN. At this point, we can deal with the most requested FQDN and see what happens when adding the less requested FQDN.

Final Evaluation

Once routing tables are built (cf. methods detailed in previous Sections 8–10), we evaluate them. A routing table takes into account only the most requested FQDN. To validate the previously built routing tables and to decide which one is the best, we perform simulations. A simulation consists in replaying the traffic on a simulator. The traffic replayed is a 10-min slot received on one of our resolving platforms at a rush hour. This allows us to perform evaluation including FQDN which are not balanced thanks to the previously built routing table. The simulator is a program implementing the load balancing task and some basic functions of the DNS servers to reproduce the behavior of a DNS resolving platform. The functions implemented are the only ones needed to evaluate performance. The indicators computed by the simulator are for each DNS server.

Network related indicators

- Query rate
- Response rate DNS-Related Indicators
- Number of signatures to be checked if DNSSEC is used
- Cache management
- CHR
- Cache length
- Number of resolutions to perform on the Internet

The result of the simulation consists of an array containing these indicators for each server of the platform. The next step is to analyze the various indicators computed from simulations. To do so, we seek for a comprehensive representation of these indicators.

To visualize the repartition of the resources over the platform and to compare the different routing tables, we use the graphical function boxplot. Handling a boxplot allows us to see immediately the median, the quartiles, and the minimum and maximum in term of resources needed by servers. Note that the median is a more interesting indicator than the mean as every FQDN-based load balancing generates exactly the same number of requests and resolutions for the whole platform. Boxplots are directly drawn thanks to the boxplot() function.

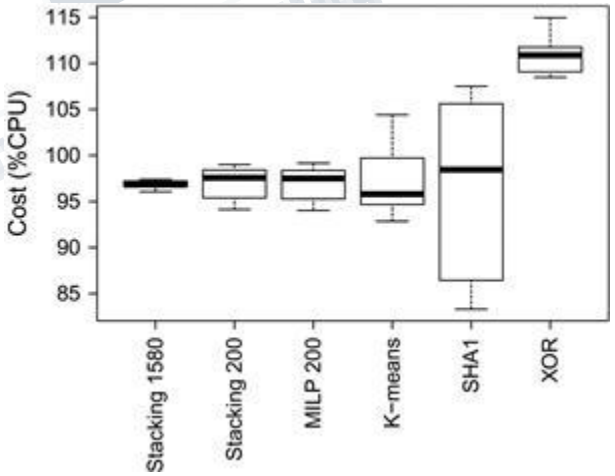

FIGURE 11 Repartition of costs.

XII CONCLUSION

R is an attractive tool to explore data and to design scripts on the basis of statistical methods. It is also efficient to visualize data. Same as python, it is useful for fast prototyping. The interactive mode allows us to find

and test different options for the different built-in functions to be included in scripts. Script mode is useful for process automation. As in python, one can take advantage of the object oriented possibilities offered by this language to ease scripts design.

R is complete with its extensions provided thanks to a variety of officially supported packages which ease its use. For common statistical-oriented usages, functions already exist. Moreover, the documentation is complete and gives us references to the algorithms the functions implement. One of the advantages of R is the possibility of making graphs with almost every R object. This is useful to visualize the effects of the processing performed on the data.

To our knowledge, FQDN-based load balancing techniques and the methods used to build the related routing tables are novel approaches to address the problem of Internet resolving platforms optimization. In the application case of data mining methods implemented in R , it was demonstrated that FQDN-based load balancing is efficient for improving the CHR and for reducing the resources needed to process DNS(SEC) traffic on a resolving platform. We can take advantage of the most popular FQDN distribution to improve this load balancing. Further works on the platform optimization problem described in this chapter include a more efficient processing of the rarely requested FQDN and the study of robustness for the proposed load balancing techniques.

REFERENCES

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

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

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

[4]. Cox TF, Cox MAA. Multidimensional Scaling. Boca Raton, FL: Chapman and Hall; 2001.

[5]. Development Core Team R. R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing; 2010.

[6]. Francfort S, Migault D, Senecal S. A bi-objective Mixed Integer Linear Program for load balancing DNS(SEC) requests. In: Proceedings of DNS EASY 2011, extended version in International Journal of Critical Infrastructure Protection, Elsevier, 2012.

[7]. Griffiths, C., 2009. Comcast DNSSEC Trail Test Bed. North American Network Operator Group (NANOG45).

[8]. Hastie T, Tibshirani R, Friedman J. The Elements of Statistical Learning: Data Mining, Inference and Prediction. In: second ed. Springer 2008.

[9]. Kogan J. Introduction to Clustering Large and High-Dimensional Data. New York: Cambridge University Press; 2007.

[10]. Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., 2005. Cluster analysis basics and extensions. Rousseeuw et al provided the S original which has been ported to R by Kurt Hornik and has since been enhanced by Martin Maechler: speed improvements, silhouette() functionality, bug fixes, etc. See the n_Changelog' file (in the package source).

[11]. Migault, D., 2010. Performance measurements on bind9/nsd/unbound. In IETF79. IEPG.

[12]. Migault D, Laurent M. How DNSSEC resolution platforms benefit from load balancing traffic according to fully qualified domain name. In: Proceedings of CSNA. 2011.

[13]. Migault D, Girard C, Laurent M. A performance view on DNSSEC migration. In: Proceedings of CNSM 2010.